NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN AND IMPLEMENTATION OF AN INERTIAL NAVIGATION SYSTEM FOR REAL TIME FLIGHT OF AN UNMANNED AIR VEHICLE

By

Dimitris Eleftherios Kataras

March 1995

Thesis Advisor: Isaac I.Kaminer
Thesis Co-Advisor: Michael K. Shields

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1995	Master's Thesis

4. TITLE AND SUBTITLE
DESIGN AND IMPLEMENTATION OF AN INERTIAL NAVIGATION SYSTEM FOR REAL TIME FLIGHT OF AN UNMANNED AIR VEHICLE

5. FUNDING NUMBERS

6. AUTHOR(S)
Kataras, Dimitris, E.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Unmanned Air Vehicles present increasing benefits to potential users, with their portability, low cost, and simplicity of operation in any battle theater. Work performed at the Naval Postgraduate School on the ARCHYTAS vehicle converges to totally unattached, autonomous flight in all regimes. Control of this inherently unstable platform is achieved by a Stability Augmentation System, designed to provide flight worthy response to pilot commands from a remote ground station. A critical part of this controller is the navigation system, which furnishes imperative data for the controlling process. This thesis examines the successful design and integration of an Inertial Navigation Sensor Suite with the already existing flight controller for the ARCHYTAS. An Inertial Measurement Unit (IMU) is selected from available systems, and the integration process is presented. The research focuses on system requirements, controller modification, IMU-controller interface procedures, and hardware configuration and installation. A radio frequency communication link that transmits data from the airborne IMU to the ground based flight controller is also developed and tested. Evaluation of the design and implementation is provided through laboratory hardware-in-the-loop tests.

14. SUBJECT TERMS	15. NUMBER OF PAGES
UAV, Inertial Navigation System, Real Time Flight Controller, Data Link, Hardware-in-the-Loop Test	186
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std 239-18

Approved for public release; distribution is unlimited.

DESIGN AND IMPLEMENTATION OF AN INERTIAL NAVIGATION SYSTEM FOR REAL TIME FLIGHT OF AN UNMANNED AIR VEHICLE

by
Dimitris E. Kataras
Lieutenant J.G., Hellenic Navy
B.S., Hellenic Naval Academy, 1987

Submitted in partial fulfillment
of the requirements for the degrees of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING
and
MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 1995

Author: _____
Dimitris E. Kataras

Approved by: _____
Isaac I. Kaminer, Thesis Advisor

M. K. Shields
Michael K. Shields, Thesis Co-Advisor

M. A. Morgan
Michael A. Morgan, Chairman
Department of Electrical and Computer Engineering

Daniel J. Collins
Daniel J. Collins, Chairman
Department of Aeronautics and Astronautics

ABSTRACT

Unmanned Air Vehicles present increasing benefits to potential users, with their portability, low cost, and simplicity of operation in any battle theater. Work performed at the Naval Postgraduate School on the ARCHYTAS vehicle converges to totally unattached, autonomous flight in all regimes. Control of this inherently unstable platform is achieved by a Stability Augmentation System, designed to provide flight worthy response to pilot commands from a remote ground station. A critical part of this controller is the navigation system, which furnishes imperative data for the controlling process. This thesis examines the successful design and integration of an Inertial Navigation Sensor Suite with the already existing flight controller for the ARCHYTAS. An Inertial Measurement Unit (IMU) is selected from available systems, and the integration process is presented. The research focuses on system requirements, controller modification, IMU-controller interface procedures, and hardware configuration and installation. A radio frequency communication link that transmits data from the airborne IMU to the ground based flight controller is also developed and tested. Evaluation of the design and implementation is provided through laboratory hardware-in-the-loop tests.

TABLE OF CONTENTS

I. INTRODUCTION 1

A. RESEARCH OBJECTIVE 2

B. PREVIOUS AND CONCURRENT RESEARCH 3

C. EXECUTIVE SUMMARY 3

II. AIRCRAFT OVERVIEW 5

A. BACKGROUND 5

B. AIRCRAFT CHARACTERISTICS 7

C. CHAPTER SUMMARY 9

III. INS/IMU SENSOR OVERVIEW 11

A. BACKGROUND 11

1. Gimbaled IMU 12
2. Strapdown IMU 13

B. ARCHYTAS SENSOR DESCRIPTION 13

C. SNL ANALOG STABILITY SUITE 15

1. Angular Rate Sensors 16
2. Vertical Gyroscope 16

D. WATSON INERTIAL MEASUREMENT UNIT (IMU-600D) 16

1. Power Requirement 19
2. Installation 19
3. Fundamentals of Operation 20
4. Rate Sensors 20
5. Accelerometers 20
6. Bank and Elevation 20
7. Altimeter Input 21

8. Output Ranges 21
9. Output Format 22
A. 'New' IMU (S.N. 24) 22
B. 'Old' IMU (S.N. 12) 24
10. RS-232 Input Commands 24
A. 'New' IMU (S.N. 24) 24
B. 'Old' IMU (S.N. 12) 26
11. Error Limiting 27
12. Initialization 27

E. CHAPTER SUMMARY 28

IV. CONTROLLER OVERVIEW 29

A. DESIGN PROCEDURE 29

B. ARCHYTAS MODEL DEVELOPMENT 34
1. Notation and Conventions 34
2. Equations Of Motion 41
3. Computer Modeling 42
4. SystemBuild Model 42
5. ARCHYTAS Model Testing 43

C. CONTROLLER DEVELOPMENT 43
1. Synthesis and Computer Modeling 43
2. Discretization and Testing 44

D. CHAPTER SUMMARY 44

V. IMU - CONTROLLER INTEGRATION 47

A. SYSTEM REQUIREMENTS 47

B. CONTROLLER MODIFICATION 48
1. 'plant_kinematics' SuperBlock 50
2. 'control_process' SuperBlock 50
3. 'imu_logic' SuperBlock 52

4. 'joystk_logic' SuperBlock 52
5. Conversion Block 53

C. INTERACTIVE ANIMATION MODIFICATION 53
1. Operation Display 53
2. Calibration Display 53

D. IMU INSTALLATION 54

E. IMU & C30 INTERFACING REQUIREMENTS 56

F. SERIAL DRIVER FOR IMU & AC100C30 INTERFACE 56
A. IMU-600D/38 and IMU-600D/42 Hexadecimal 58
B. IMU-600D Binary 58

G. ARCHYTAS SAS SETUP 60

H. CHAPTER SUMMARY 66

VI. SYSTEM EVALUATION AND RESULTS 67

A. CLOSED-LOOP TESTING 67
1. Setup 68
2. SystemBuild Testing Results 69
3. AC100 Model C30 Testing Results 69

B. HARDWARE-IN-THE-LOOP TESTING 70
1. Setup 70
2. HITL Results 73

C. FLIGHT TEST 74

D. CHAPTER SUMMARY 76

VII. DATA LINK DEVELOPMENT 77

A. OVERVIEW 77

B. REQUIREMENTS 78

C. PREVIOUS RESEARCH 79

D. DEVELOPMENT 80
1. SLQ-96 General Information 80
A. Description 80
B. Radio Data Transfer 81
C. Data Transfer Reliability 81
D. Half- or Full-Duplex Operation 82
E. Interfacing and Compatibility 82
F. Mechanical & Electrical Specifications 83
2. Datalink Setup 83
3. SLQ-96 Configuration 83
4. Operation 85
E. EVALUATION 85
1. Setup 86
2. HITL Results 88
F. CHAPTER SUMMARY 88
VIII. COMMENTS AND CONCLUSIONS 89
A. ACCOMPLISHMENTS 89
B. RECOMMENDATIONS 91
1. System Tests 91
A. Test Stand 91
B. Tethered Flight 92
C. Actual Flight 93
2. SAS Software Recommendations 94
3. SAS Hardware Recommendations 94
4. General System Recommendations 95
C. SUMMARY 96
APPENDIX A. MATRIX$_X$ - SystemBuild BLOCK DIAGRAMS 97

APPENDIX B. SERIAL DRIVERS FOR IMU & AC100C30 INTERFACE 131

A. SERIAL DRIVER DEVELOPMENT 131
1. The 'get_SERIAL_parameters' Function 131
2. The 'user_SERIAL_out' Function 132
3. The 'user_sample_SERIAL_in' Function 134

B. IMU OUTPUT CONVERSION 135

C. SERIAL DRIVERS FOR THE IMU 136
1. Serial Code Part Provided By Isi 136

D. CODE LISTING FOR HEX OUTPUT MODES 143

E. CODE LISTING FOR BINARY OUTPUT MODE 149

APPENDIX C. FLIGHT TEST EQUIPMENT SETUP PROCEDURE 155

A. DIRECTORY STRUCTURE 155

B. MOVING PROCEDURE 156

APPENDIX D. SLQ-96 SPECIFICATIONS AND OPERATION 157

A. SPECIFICATIONS 157

B. OPERATION AND CONFIGURATION 158
1. Input/Output Signals 158
2. Status Lights 159
3. Dip Switch Settings 160

LIST OF REFERENCES 165

INITIAL DISTRIBUTION LIST 167

LIST OF FIGURES

2.1: AIRBORNE REMOTELY OPERATED DEVICE, AROD 6

2.2: ARCHYTAS DIRECTION OF POSITIVE VANE DEFLECTIONS 8

3.1: GIMBALLED IMU 12

3.2: ARCHYTAS BODY COORDINATE SYSTEM 15

3.3: WATSON IMU-600D INERTIAL MEASUREMENT UNIT 17

3.4: IMU to RS-232C (NINE-PIN) CONNECTIONS 18

4.1: COORDINATE SYSTEMS RELATIVE POSITION 34

5.1: ARCHYTAS SAS HIGHEST LEVEL 49

5.2: OPERATION IA DISPLAY 54

5.3: CALIBRATION IA DISPLAY 55

5.4: ARCHYTAS BODY AND INERTIAL COORDINATE SYSTEMS 56

5.5: IMU - WIRING HARNESS TETHER CONNECTIONS 57

5.6: SERIAL DRIVER LOGIC FLOWCHART 59

5.7: ARCHYTAS HARDWARE SETUP, TETHERED FLIGHT 60

5.8: CONNECTOR END OF WIRING HARNESS TETHER 62

5.9: AC100 MODEL C30 HITL TEST WIRING DIAGRAM 63

5.10: WIRING HARNESS - C30 I/O CONNECTIONS (PORT A) 64

5.11: WIRING HARNESS - C30 I/O CONNECTIONS (PORT B) 65

6.1: CLOSED-LOOP BLOCK DIAGRAM 68

6.2: ARCHYTAS HITL SETUP, TETHERED FLIGHT 72

6.3: FLIGHT TEST SETUP 75

7.1: UNTETHERED MODE CONFIGURATION 78

7.2: DOWNLINK SETUP 84

7.3: TRANSMITTING END CONNECTOR CABLE 85

7.4: RECEIVING END CONNECTOR CABLE 86

7.5: ARCHYTAS HITL SETUP, UNTETHERED FLIGHT 87

8.1: FLIGHT TEST SETUP 92

8.2: TEST STAND 93

8.3: ACTUAL FLIGHT SETUP 94

A.1: 'actuators' SUPERBLOCK 100

A.2: 'actuator_1' SUPERBLOCK 101

A.3: 'ang_velocity_eq' SUPERBLOCK 102

A.4: 'control_process' SUPERBLOCK 103

A.5: 'dcont_wind' SUPERBLOCK 104

A.6: 'feedback_laws' SUPERBLOCK 105

A.7: 'filters' SUPERBLOCK 106

A.8: 'flight_test_1' SUPERBLOCK 107

A.9: 'imu_in' SUPERBLOCK 108

A.10: 'imu_logic' SUPERBLOCK 109

A.11: 'input_to_model' SUPERBLOCK 110

A.12: 'input_to_vane_servos' SUPERBLOCK 111

A.13: 'integ_ang_vel' SUPERBLOCK 112

A.14: 'integ_lin_vel' SUPERBLOCK 113

A.15: 'integ_sim' SUPERBLOCK 114

A.16: 'int_ang_sim' SUPERBLOCK 115

A.17: 'joystk' SUPERBLOCK 116

A.18: 'joystk_logic' SUPERBLOCK 117

A.19: 'kinematics' SUPERBLOCK 118

A.20: 'L_dot_eq' SUPERBLOCK 119

A.21: 'l_m_n_compute' SUPERBLOCK 120

A.22: 'lin_velocity_eq' SUPERBLOCK 121

A.23: 'pitch_command' SUPERBLOCK 122

A.24: 'plant_kinematics' SUPERBLOCK 123

A.25: 'rollrate_command' SUPERBLOCK 124

A.26: 'rpm_command' SUPERBLOCK 125

A.27: 'T_value' SUPERBLOCK 126

A.28: 'vane1_processing' SUPERBLOCK 127

A.29: 'vane4x' SUPERBLOCK 128

A.30: 'yaw_command' SUPERBLOCK 129

LIST OF TABLES

2.1: PHYSICAL CHARACTERISTICS OF ARCHYTAS 7

2.2: VANE DEFLECTION COMBINATIONS FOR POSITIVE ANGLES 8

3.1: IMU-600D NINE-PIN MALE CONNECTOR 18

3.2: IMU OUTPUT RANGES 21

3.3: DATA BANK ONE 25

3.4: DATA BANK TWO 25

D.1: SLQ-96 I/O SIGNALS 158

D.2: DIP SWITCH FUNCTIONS 161

D.3: DATA RATE SELECTION 162

D.4: THREE-WIRE TIMEOUT SELECT 162

D.5: CTS DELAY SELECT 162

D.6: CHARACTER LENGTH SELECT 163

ACKNOWLEDGMENT

I would like to express my gratitude and appreciation for the assistance provided in the development of this thesis by Dr I. I. Kaminer, LCDR M. K. Shields and Dr R. M. Howard. The completion of this project relied on their guidance, support and professional counsel.

I. INTRODUCTION

Unmanned Air Vehicles (UAVs) can offer many benefits to potential users, such as relatively low procurement cost, low cost per flight hour, portability and simplicity of operation. UAV's provide a cost-effective and fatigue-resistant solution to many airborne missions, such as Search and Rescue, Over the Horizon Early Warning and Targeting, Coastal Patrol and Area Surveillance, with a low profile and low electromagnetic (EM) emission platform. Today, research in UAVs is at the forefront of technology. The goal of the UAV development project at the Naval Postgraduate School is totally unattached, autonomous flight in all regimes - automatic take-off, automatic waypoint tracking, and finally automatic landing. In the future, capability of automatic transition from vertical to horizontal flight will be examined.

The ARCHYTAS UAV being developed at the Naval Postgraduate School is completely dependent upon its automated Stability Augmentation System (SAS) to provide control of the aircraft in flight. In the center of the SAS is a microprocessor based flight controller. The correct functioning and real-time coordination of all processes on board the aircraft depends on their interaction with this controller.

A critical part of the SAS is the navigation system which provides inputs to the processor that are imperative to the controlling process. The kind of navigation system used, and its characteristics, depend on the nature of the controller itself. Properties such as update rate and accuracy are of great significance. The most popular long range navigation system in use is the Inertial Navigation System (INS). At present, the UAV's primary means of guidance is based on inertial navigation provided by an onboard sensor package, the Inertial Measurement Unit (IMU). Future plans include integration of an onboard Global Positioning System (GPS) receiver, and other in-flight sensors. Appropriate data must be selected from this navigation suite and analyzed to determine the states of the aircraft at any given time. These states may have to be converted into a different coordinate system. Processing the states via appropriate control algorithms yields corrective deflections for the available control surfaces and the throttle. These control

surfaces are moved by pulse-width modulated servos, which require a pulse of a specified width to be generated and output at a certain time. Control surface deflections are sent to the vehicle and aircraft states are received by the controller through the communication link, with the appropriate protocols. Device driver programs control inputs and outputs (I/O) and provide interfacing between different components of the UAV. All of these operations are repeated at various intervals and coordinated through the already developed, real-time controller.

A. RESEARCH OBJECTIVE

The goal of this research is to design an INS Suite, and implement it with a real-time controller for the ARCHYTAS UAV. The INS will consist of an IMU, which will be positioned onboard the vehicle. The IMU's basic function will be to provide attitude, rate and position values to the controller, in an appropriate form.

This research addresses the following questions:

- What hardware components are available for the task?
- What is the form and limitations of the data provided by the available INS sensors?
- What is the required rate and bandwidth for angular position by the existing controller?
- What is the format of the data required by the existing controller software?
- What other I/O is required?
- How can the INS/IMU hardware be integrated on the UAV?
- What hardware is necessary to perform this functionality?
- How will the aircraft communicate with the ground control station?
- How well does the IMU-controller combination perform in the lab tests and actual restrained flight?

- What are the necessary developments for a wireless communication link?
- What are the format and buffering requirements for this communication link?

These tasks require modification of the existing flight controller to facilitate integration with the IMU module. After successful design, development and implementation, the integrated SAS will be tested, with its hardware installed in working order onboard the UAV. A radio frequency (RF) data link will also be developed, for wireless transmission of data from the airborne IMU to the ground based flight controller. Testing of the overall system will be performed initially during restrained tether-connected flight, while successful remote, radio-controlled free flight is the ultimate future goal.

B. PREVIOUS AND CONCURRENT RESEARCH

The ARCHYTAS UAV project under development at NPS is the culmination of several previous and concurrent research programs. Much progress has already been achieved on this area, and some of these questions have already been covered. Previous thesis research on the UAV will be combined and adapted to the present stage of the project. The airframe was aerodynamically analyzed and modified by Stoney [Ref. 1]. The analog sensors available for Inertial Navigation and the ARCHYTAS actuator servos were discussed by Moran [Ref. 2]. The control algorithm has been developed by Sivashankar [Ref. 3] and Kuechenmeister [Ref. 4]. The SystemBuild controller project currently used has been implemented and tested by Moats [Ref. 5]. Development of serial drivers for interface of the controller were covered by Noyes [Ref. 6]. Work on the design of a real-time controller for the UAV has also been done by Hoffman [Ref. 7]. Two datalinks have been developed to facilitate the transfer of data to and from a ground station. Reichert [Ref. 8] designed a wide-band UHF system and Bess [Ref. 9] tested a spread spectrum datalink.

C. EXECUTIVE SUMMARY

This report consists of eight chapters, including this introduction. Three background chapters are provided to familiarize the reader with the UAV project. A

generalized description of the platform is presented in Chapter II. Aircraft components and characteristics are also discussed. In Chapter III a review of Inertial Navigation Systems is included, and the available Inertial Measurement Units are described and compared. An overview of the existing Stability Augmentation System is included in Chapter IV. Description of the platform model development and the controller design are included to provide insight into the controller architecture and operation.

Requirements, considerations and modifications for the integration of the controller with the IMU are discussed in Chapter V. System requirements are presented, along with essential hardware and software modifications. This chapter includes detailed discussion of IMU installation and implementation, desired data and IMU output format, and the final ARCHYTAS SAS setup. The necessary hardware connections, data format and flow, and required I/O interface drivers are also presented. Chapter VI deals with evaluation and testing of the final IMU-Controller implementation. Closed-loop controller testing and the required process for a Hardware-in-the-Loop (HITL) test are discussed. The corresponding setup and results are also provided. Chapter VII contains the requirements for, and the development of, a radio frequency (RF) down link for the transmission of IMU data to the ground based controller. Conclusions and accomplishments of this study, along with recommendations for improvement of the UAV project are included in Chapter VIII. A procedure for flight evaluation in a test stand as well as steps for tethered and wireless flight testing are also provided. A detailed description of the SystemBuild project for the ARCHYTAS SAS is presented in Appendix A. Appendix B discusses the development of the serial drivers that interface the controller with the digital IMU, and contains the drivers used in the project. Appendix C provides instructions for the field equipment setup, which will be used during flight testing. Appendix D includes the specifications and operation procedures for the developed RF down link.

II. AIRCRAFT OVERVIEW

The current UAV research at NPS has focused on the vertical take-off and landing (VTOL) ARCHYTAS UAV, as the platform for modeling, development and testing of various guidance, navigation and control techniques and systems. This chapter provides the reader with a general overview of the platform used for the ARCHYTAS UAV. It contains a description of the project development and includes the aircraft characteristics.

A. BACKGROUND

The ARCHYTAS, named after a Greek contemporary of Plato who was credited with designing and successfully flying a mechanical bird, is serving as a testbed for propulsion and stability work leading to the design and construction of the full scale demonstration vehicle. The ARCHYTAS is based on the Airborne Remotely Operated Device (AROD) aircraft. Originally designed in the mid-1980's by the Sandia National Laboratories (SNL) in Albuquerque, New Mexico, in conjunction with the Naval Ocean System Command, the AROD is a VTOL vehicle [Ref. 2]. It was developed to meet the U.S. Marine Corps requirement for an intelligence gathering UAV [Ref. 10, 11]. The vehicle as originally designed, is shown in Figure 2.1.

The AROD was built to fly strictly in a hover or vertical flight mode. While filling the requirement for operation from unprepared areas, the configuration was inefficient for transitional flight by the nature of its vertical flight mode. However, its ducted-propeller design did offer the advantage of safety, when operated in close proximity to ground personnel, as well as high thrust efficiency. The vehicle was intended to be controlled primarily through a fiber-optic link, with RF control available for backup and as a training aide [Ref. 10]. It first flew successfully in 1986, but never went into production. One of the reasons the AROD was abandoned, was severe vibration during operation. Also, an updated requirement by the newly-formed UAV Joint Project Office (UAV JPO), called for the UAV to be able to achieve translational flight, rather than just hovering flight. The project was canceled in 1987 due to budget cutbacks. Equipment assets remaining from

the AROD program were acquired by NPS in 1992, and are used as the base for the ARCHYTAS aircraft, in multiple research fields.

In previous research papers [Ref. 2, 8] the name ARCHYTAS had been used for a project to design a UAV capable of hovering and transitional flight. That platform consisted of the AROD body forming the fuselage, and wings that were attached to it, in order to provide lift during horizontal flight. At present this project is delayed, and the name ARCHYTAS refers to a vehicle capable of vertical flight only.

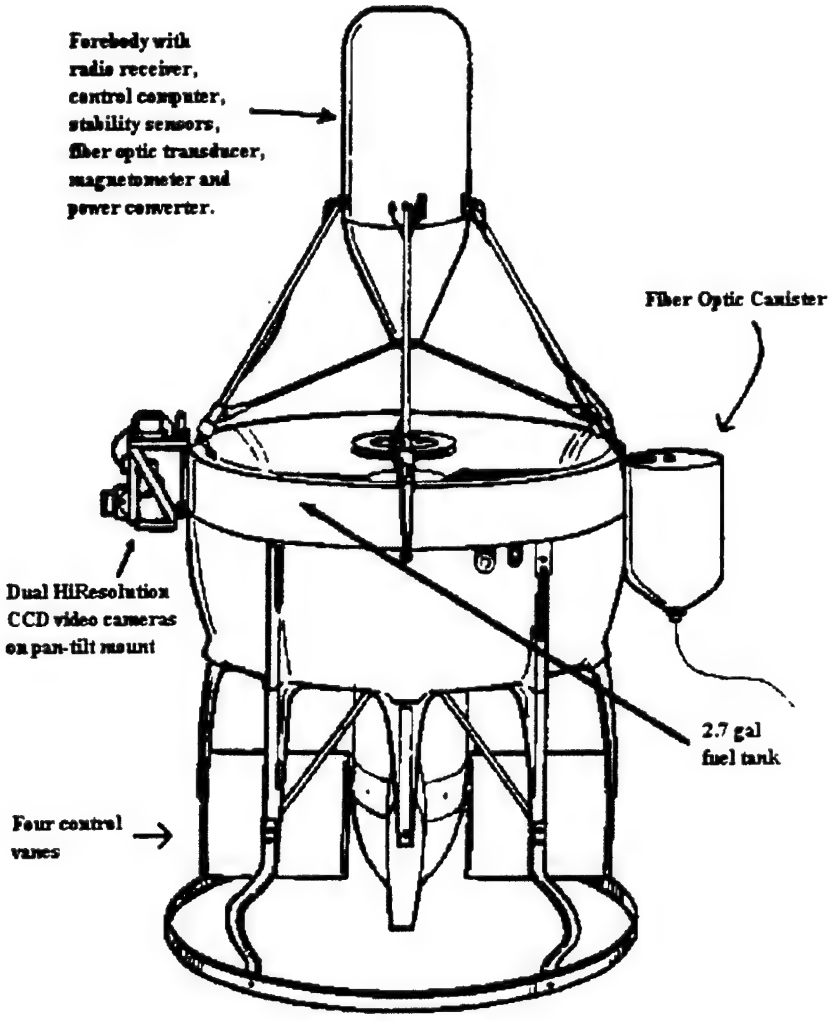


Figure 2.1: Airborne Remotely Operated Device, AROD [Ref. 11]

B. AIRCRAFT CHARACTERISTICS

The main features of the ARCHYTAS are vertical take-off and landing, lightweight construction, compact size, and minimal support equipment required [Ref. 4]. However, the vehicle requires most of the engine output to maintain powered lift, so very little excess thrust is left for translational flight. Control of the aircraft is obtained through the use of four fixed anti-torque vanes and four moveable control vanes positioned in the propeller wash of the duct. The vehicle characteristics are presented in Table 2.1.

Inlet Diameter, A	29.25 in
Propeller Radius, R	12 in
Exit Radius	23.375 in
Inlet Area Ratio	1.22
Exit Area Ratio	1.12
Exterior Contour	Tapered Rear
Propeller Location % chord	25 %
Number of Blades	3
Engine Speed, Max.	8000 rpm
Engine Speed, Min.	6500 rpm
Tip Speed, Max.	838 fpm
Tip Speed, Min.	680 fpm
Power Loading	7.25 HP/f^2
Mass Moment of Inertia, I_x	1.2312 slug - f^2
Mass Moment of Inertia, I_y	3.9548 slug - f^2
Mass Moment of Inertia, I_z	3.9825 slug - f^2
Propeller Mass Moment of Inertia, I_{rx}	0.00898 slug - f^2
Propeller Mass Moment of Inertia, I_{ry}	0.0045 slug - f^2
Propeller Mass Moment of Inertia, I_{rz}	0.0045 slug - f^2

Table 2.1: Physical Characteristics of ARCHYTAS

The ARCHYTAS is powered by a Herbandson Engines, Inc., Dyad 280. This is a 26-horsepower, 2-stroke, 2-cylinder gasoline engine, derived from a commercial chainsaw motor. Coupled with a three-bladed ducted propeller, it provides approximately 120 lb. of maximum thrust. The moveable control vanes are all used in combination to exert the desired control forces on the vehicle. Roll control is achieved by deflecting the four vanes in the same direction, while pitch and yaw control are obtained by deflecting the pair of vanes in the y and z planes respectively. The numbering of the vanes is shown in Figure 2.2 and the combinations of vane deflections required for positive roll, pitch, and yaw motion are tabulated in Table 2.2.

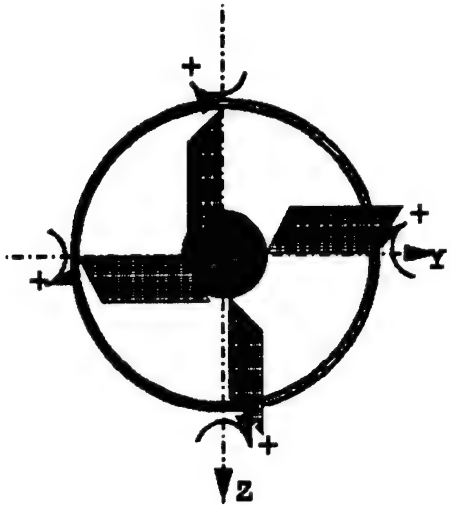


Figure 2.2: Archytas Direction of Positive Vane Deflections [Ref. 11]

	Vane Combination
Roll, Φ	$V_1 + V_2 + V_3 + V_4$
Pitch, Θ	$V_2 - V_4$
Yaw, Ψ	$V_1 - V_3$

Table 2.2: Vane Deflection Combinations for Positive Angles

An important aspect of the ARCHYTAS design is the improvement in static performance provided by the efficiency of the ducted fan design. The addition of the shroud around the three-bladed propeller resulted in increased mass flow through the fan, and thus more static thrust when compared to a conventional propeller configuration [Ref. 12]. The use of a single propeller or fan unit in a ducted-fan system simplifies the design, but requires a stability system to counteract the torque and to account for the gyroscopic coupling. There are three dynamic coupling effects which were considered when designing a control system for the ARCHYTAS [Ref. 3, 4]. The first is the gyroscopic coupling due to the large angular momentum of the propeller. Another dynamic coupling exists between the vehicle attitude and the attitude-rate since thrust vectoring is required for translational movement. The third effect is caused by the propeller. As the air is accelerated through the propeller, it is also deflected slightly, causing a swirl effect. The result is that the air mass strikes the control vanes at an angle proportional to the angular rate of the propeller. This creates a rolling moment which is dependent on the throttle input.

C. CHAPTER SUMMARY

The ARCHYTAS VTOL aircraft is the focal point of the NPS UAV project. It exhibits unique characteristics that were taken into account in the design of the flight controller for the ARCHYTAS Stability Augmentation System (SAS), and in selecting an INS for the UAV. Detailed discussions are included in Chapters III and IV.

III. INS/IMU SENSOR OVERVIEW

This Chapter begins with a brief description of Inertial Navigation Systems (INS), theory and the two conceptual methods of INS implementation. It follows with a detailed presentation of the INS sensors available for the ARCHYTAS. Both the AROD analog stability suite and the digital Watson IMU are covered.

A. BACKGROUND

Inertial Navigation has long been the standard for self-contained, long-range aircraft navigation.

> Inertial Guidance or Navigation is the process of directing the movement of a rocket, ship, aircraft, or other vehicle from one point to another, which is based on sensing acceleration of the vehicle in a known spatial direction, by instruments which mechanize the Newtonian laws of motion, and integrating acceleration to determine velocity and position. [Ref. 13]

An INS senses aircraft thrust acceleration, angular rates and spatial orientation and computes the inertial acceleration. This acceleration can then be integrated to find velocity and position.

The primary component in any Inertial Navigation System is the Inertial Measurement Unit (IMU). The IMU is composed, in general, of three accelerometers, three rate gyros, and two inclinometers. The accelerometers measure thrust acceleration. Thrust acceleration is composed of linear, centripetal, and gravitational effects. Einstein's Principle states that it is impossible for a sensor to distinguish between the effects of gravity and acceleration. Thus, the thrust acceleration that it provides, is given by the equation

$$^{B}\alpha = {}^{B}\dot{v} + {}^{B}g \tag{3.1}$$

where $^{B}\alpha$ is the vehicle acceleration, $^{B}\dot{}$ is the inertial acceleration, ^{B}v is the velocity and ^{B}g is gravity, expressed using avionics design notation [Ref. 14]. All quantities are in body coordinates.

Rate gyros sense angular velocities. These angular velocities can be resolved in either the body or inertial coordinates, depending on the type of IMU implementation. The two inclinometers sense aircraft attitude, i.e., bank and elevation angles.

There are two conceptual methods for implementing Inertial Navigation:

- Gimballed IMU
- Strapdown IMU

Short descriptions of each method, are presented below.

1. Gimbaled IMU

A Gimballed IMU rotates about its four gimbals during operation, as shown in Figure 3.1.

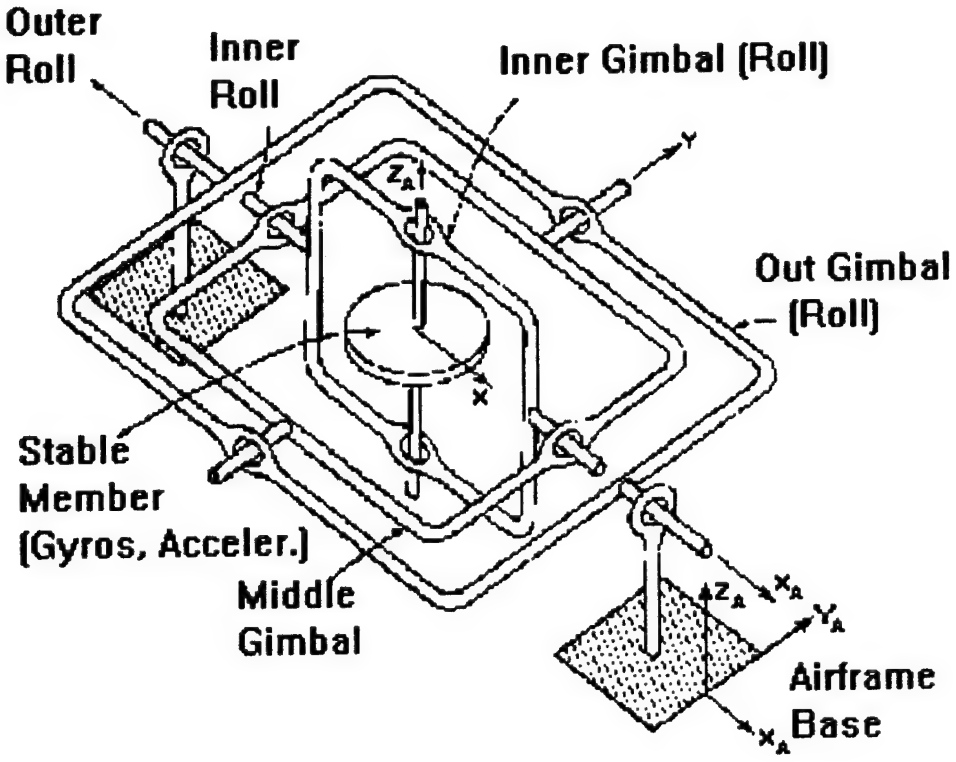


Figure 3.1: Gimballed IMU [Ref. 15]

Aircraft IMU's must have four gimbals to prevent gimbal lock, while earthbound IMU's require only three gimbals [Ref. 15]. A controller maintains the IMU in a constant

orientation toward true North and lying in the locally horizontal plane. By maintaining this orientation, the gimballed IMU measures inertial quantities directly. This data can be integrated without transformation to yield inertial velocity and position. From a navigation standpoint, this configuration seems ideal. However, gimballed systems are large and heavy, making them impractical for small aircraft. Also, kinematic quantities resolved in the aircraft-fixed coordinate system, necessary for stability augmentation and control, are not directly available. Instead, they must be computed through a series of Euler rotations. The computations required, take time and thus introduce delays into the time critical control problem. These facts make the gimballed system unsuitable for the ARCHYTAS.

2. Strapdown IMU

Strapdown IMU is conceptually the reverse of the *Gimballed* system described above. Rather than maintaining a constant inertial orientation, a strapdown system is "strapped down" to the aircraft, thus maintaining a constant orientation in the aircraft-fixed coordinate system. Therefore, the output of the IMU is resolved in the local coordinate system. Kinematic quantities are immediately available for the control system. The extra computational burden now rests on the navigation computer which must transform the accelerations and angular velocities sensed in the local coordinate system to the inertial system. Currently, most inertial systems are strapdown, due to the advent of high speed, low cost, lightweight computers.

B. ARCHYTAS SENSOR DESCRIPTION

The flight parameters needed to be monitored and fed back for the ARCHYTAS Stability Augmentation System (SAS) to operate in a vertical hover flight mode, have been determined to be angular rotation rates in roll (p), pitch (q) and yaw (r), about each body axis x, y, and z respectively [Ref. 14], as well as pitch (θ) and yaw (ψ) angles. To provide these necessary parameters to the flight controller, an INS has to be integrated with the SAS. Taking into consideration the narrow weight and power margins of the

ARCHYTAS UAV, a choice can be made only among the more lightweight strapdown INS.

Currently there are two such INS suites available for the ARCHYTAS. One is the analog system used by Sandia National Laboratories (SNL), for control of the original AROD. The second is an off-the-shelf digital INS selected by the NPS project group, for integration with the new flight controller for the ARCHYTAS. Both systems are based on the same concept, but differ significantly in the fundamentals of operation.

The AROD, as originally designed and produced by SNL, used three angular rate sensors and a vertical gyroscope to provide the necessary flight parameters. These analog sensors were housed in the electronics compartment, or forebody of the AROD (see Figure 2.1). This sensor package presents the advantage of being actually tested in flight by SNL, but is not convenient for completely wireless flight, since the necessary analog signal outputs have to be transmitted on a telemetry downlink. The analog INS suite is also less expensive and easily replaceable, since there exists an inventory of AROD analog sensors, at NPS.

The digital solid state Inertial Measurement Unit (IMU), was selected by the NPS UAV group to provide the necessary angles and rates to the controller via a serial digital port. As described in Chapter V, the controller has been configured and tested for proper Hardware-In-The-Loop ground operation with this IMU. This digital INS suite has not yet been tested in flight, but the RS232 serial format of the outputted sensor signals can be easily transmitted on a single downlink channel. Unfortunately only two digital IMU's have been purchased, and their very high replacement cost dictates that they should be used with great caution during the early stages of flight testing. Once the risk is reduced and the reliability of the ARCHYTAS-controller system is proven through extensive testing, the digital IMU will become a permanent component of the flight controller .

For completeness, both Inertial Navigation suites will be described in the sections that follow. During the early stages of flight testing and evaluation either system can be integrated with the controller. Should the project team decide to replace the digital IMU

with the analog sensors, it would be necessary to modify appropriately, the controller I/O interfaces and drivers discussed in Chapter V.

C. SNL ANALOG STABILITY SUITE

SNL designers referenced AROD parameters using strut three as the 'top' of AROD [Ref. 2]. This convention led to labeling vane three, located below strut three, the 'upper rudder'. In the same sense, vane one is the 'lower rudder', vane four is the 'left elevator', and vane two is the 'right elevator'. The natural body axes as defined by SNL are then, x_B positive aligned with AROD's vertical axis and directed upward, y_B positive toward strut two, and z_B positive toward strut three [Ref. 2]. The body coordinate system for the ARCHYTAS is shown in Figure 3.2. This nomenclature is used throughout this thesis, wherever body axes are concerned.

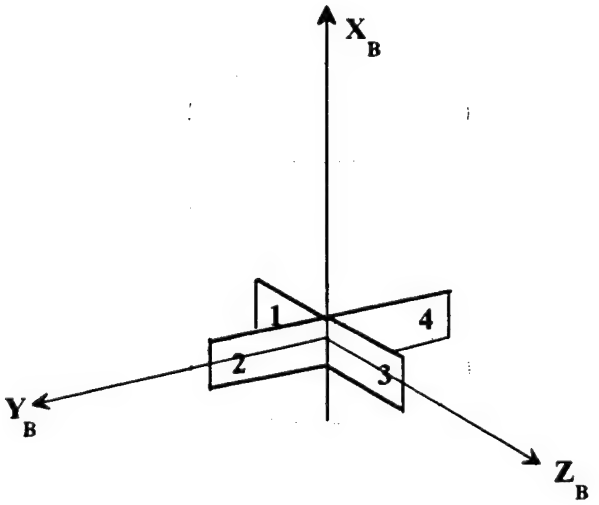


Figure 3.2: ARCHYTAS Body Coordinate System

1. Angular Rate Sensors

The angular rate sensors used aboard the AROD were Humphrey RT-01 and RT-09 single and dual rate sensors, respectively. Rate sensors are the solid state functional equivalent of a rate gyroscope. They provide an output voltage which is linearly proportional to the angular rate of the sensor [Ref. 2]. The RT-01 was used to sense roll rate about the x axis. The RT-09 measured pitch and yaw rates about the y and z axes. Moran [Ref. 2] provides diagrams of the electrical connections for the RT-01 and RT-09 angular rate sensors. His report also contains the manufacturer's specifications for both rate sensors, yielding maximum rates of 100°/sec for roll from the RT-01, and 100°/sec for pitch and yaw from the RT-09. The specifications call for at least one minute of warm-up before accurate rates can be measured. Electrical interconnections for both sensors are described in the AROD circuit diagrams provided by Moran [Ref. 2].

2. Vertical Gyroscope

To measure pitch and yaw angles from vertical, SNL used a Humphrey VG-34 vertical gyroscope. Moran [Ref. 2] includes an electrical connection diagram for the VG-34 and also provides the manufacturer's specifications for the VG-34, which show a maximum angle of +/-60° in pitch and +/-90° in yaw, each accurate to +/-1°. As listed in the specification sheet, a warm-up time of at least 5 minutes is required before accurate angles will be measured. Electrical interconnections for the vertical gyroscope are shown on the AROD circuit diagrams provided by Moran [Ref. 2].

D. WATSON INERTIAL MEASUREMENT UNIT (IMU-600D)

The Watson Inertial Measurement Unit IMU-600D was selected by the NPS UAV group to be incorporated in the ARCHYTAS Stability Augmentation System. The IMU-600D, shown in Figure 3.3, is a multi-purpose sensor package providing precision multi-axis information. Watson Industries produces this solid state gyro system which uses a microprocessor to integrate angular rate sensor data and provides a closed loop system for error correction to adjust biases from earth rotation and instrument offsets. Interface is

done through a 16 bit A/D, using a nine pin male connector. Pins are connected according to Table 3.1. In order to connect the IMU-600D to a RS-232C serial port of a PC terminal, the 9-pin male should be connected as shown in Figure 3.4, for a 9-pin standard RS-232C plug.

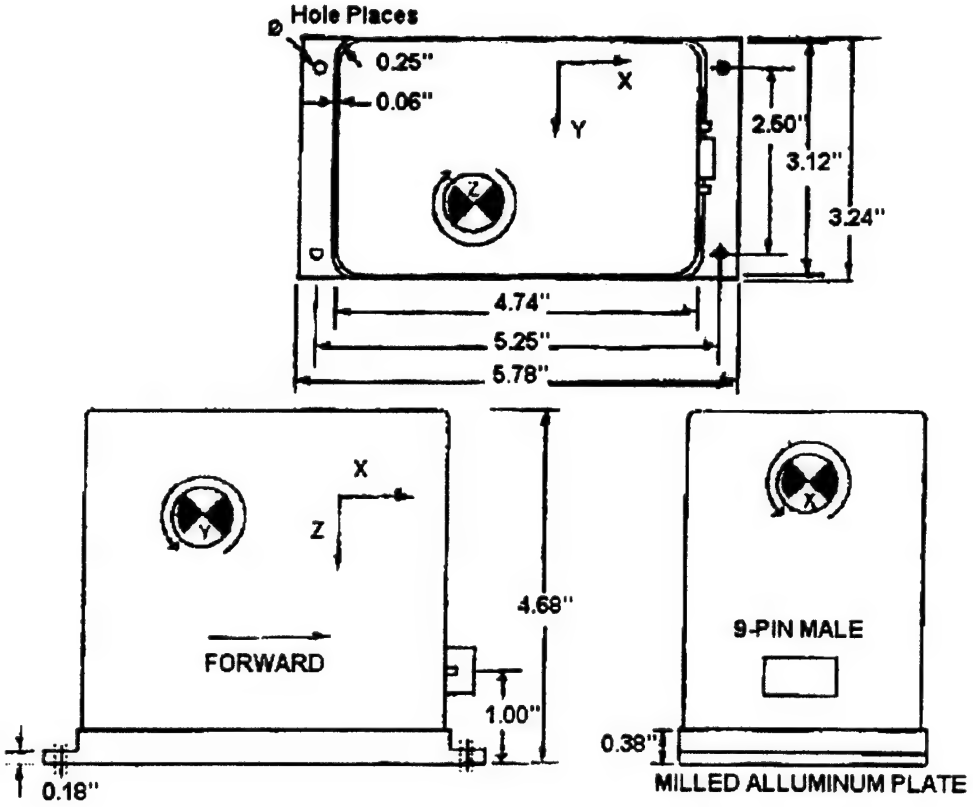


Figure 3.3: Watson IMU-600D Inertial Measurement Unit [Ref. 16]

The solid state vibrating element angular rate sensor, which is also produced by Watson Industries, is used in this system to provide high reliability, low power consumption, and low weight. In addition, this package provides three accelerometer outputs for each of the three axes.

Pin Number	Connection
1	Power Ground
2	+/- 90°
3	Signal Ground
4	-
5	Rx (from user)
6	-
7	Altimeter Input
8	-
9	Tx (to user)

Table 3.1: IMU-600D Nine-Pin Male Connector

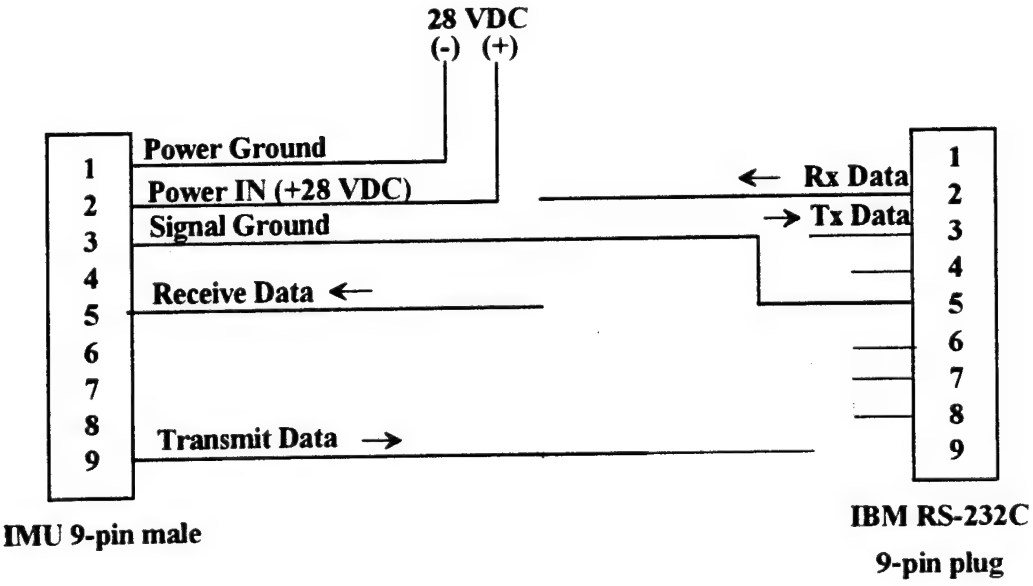


Figure 3.4: IMU to RS-232C (Nine-Pin) Connections

Currently there are two Watson IMU-600D units at NPS, which look identical but have certain differences in their features and operation. The IMU with Serial Number (S.N.) 12 was purchased first, is configured with 'old' software, and produces only hexadecimal ASCII. The unit with S.N. 24 contains 'newer' software and outputs data in decimal, hexadecimal, or binary format. In the description that follows, the two units will be covered separately where differences in characteristics and operation exist.

1. Power Requirement

The IMU-600D requires a +28VDC supply, although the unit will operate down to +22VDC. Internally, the unit converts +28VDC to +/- 15VDC and +/- 5VDC regulated supplies. Power line noise is not critical; however, a high quality electrical ground was provided.

2. Installation

This device utilizes an earth's magnetic field orientation sensor, so orientation is important. The base plate of the transducer is intended to be mounted on top of a horizontal surface in the forebody of the ARCHYTAS (see Figure 2.1), with the connector toward the forward direction of the vehicle (strut three).

A milled aluminum mounting plate is provided for a flat surface mount. The unit may be adhesively mounted at any of its surfaces. Since high shock loads are not expected (greater than 20G or repeated shocks greater than 10G), the appropriate shock mounting will not be used. It will be determined during flight testing whether vibration isolation should be provided. The unit is epoxy sealed and will provide protection in most environments.

High level AC magnetic fields, such as from large transformers, motors, or soldering guns, are potentially damaging to the circuitry, and should be avoided. Due to its size and power the ARCHYTAS motor is not expected to create any malfunctions. Exposure to high DC magnetic fields are to be avoided since this can produce a lingering self-magnetization of the sensor, which can cause distortion of the heading reference.

3. Fundamentals of Operation

The angular rate sensor signals are coordinate transformed and then integrated to produce attitude and heading outputs that reflect normal aircraft attitude coordinates. These attitude and heading signals are compared against two vertical reference pendulums and a triaxial fluxgate magnetometer to derive short term absolute errors. These errors are filtered over a long time constant and are used to adjust biases in the system, so that the long-term convergence of the system, is to the vertical references and the magnetic heading [Ref. 16].

4. Rate Sensors

The rate sensors are corrected internally for bias, gain and cross-axis errors. All sensor outputs are available on the analog channels and on the RS-232 port. The resolution of the RS-232 decimal output is 0.1°/ second.

5. Accelerometers

The accelerometers are corrected internally for bias, gain and cross-axis errors. The x, y, and z accelerometers are referenced to the actual installation of the accelerometers on the vehicle. The current inertial coordinate orientation is covered in Chapter V.

6. Bank and Elevation

The bank and elevation data is based upon a combination of pendulum and angular rate inputs. The angular rate sensors are integrated to provide accurate short-term information and the pendulums are used as long-term reference values. The full scale range of the bank is +/- 179.9° (+/-60° for S.N. 12); however, stated accuracies apply only to +/- 30°. The full scale range of the elevation is +/- 89.9° (+/-60° for S.N. 24), although stated accuracies apply only to +/- 30°. The unit will continue to provide information at all altitudes. Extended maneuvering at extreme attitudes will cause accuracy to degrade over time until more normal attitudes are reached.

7. Altimeter Input

There is capability for an external user provided analog input. This is sampled by an 8-bit ADC and is defined as the altimeter input, with a range of 0-5VDC. The data is then transmitted on the RS-232 output. The altimeter input is on pin 7 of the IMU connector.

8. Output Ranges

The factory set (F.S.) output ranges of the IMU-600D are shown in Table 3.2, for both available units. The first nine angles, accelerations and rates, are provided automatically by the IMU. The altimeter output is a value based on an analog input to the IMU-600D, provided by an external source (altimeter) to the unit. Analysis of these values reveals that, they extend well beyond the anticipated flight envelope of the ARCHYTAS.

Output	Range (S.N. 24)	Range (S.N. 12)
Bank, Φ - (F.S.)	+/- 180°	+/- 60°
Elevation, Θ - (F.S.)	+/- 90°	+/- 60°
Heading, Ψ - (F.S.)	0-360°	0-360°
x Axis Acceleration - (F.S.)	+/- 10 g's	+/- 3 g's
y Axis Acceleration - (F.S.)	+/- 10 g's	+/- 3 g's
z Axis Acceleration - (F.S.)	+/- 10 g's	+/- 3 g's
Roll Rate, p - (F.S.)	+/- 100°/s	+/- 100°/s
Pitch Rate, q - (F.S.)	+/- 100°/s	+/- 100°/s
Turn Rate, r - (F.S.)	+/- 100°/s	+/- 100°/s
Altimeter Input	0-5 VDC	Not Available

Table 3.2: IMU Output Ranges

9. Output Format

The Watson IMU-600D provides four types of sensor data: angular rates, angular positions, linear accelerations, and magnetic heading. The exact format of the output, depends on the unit being used. The data is output to the user through the nine pin male connector on the casing (Figure 3.3 and Table 3.1). Verification of the output can be performed by connecting the IMU to the serial port of a PC and running a commercial terminal or MODEM management program, such as the terminal tool of Microsoft Windows. For correct reception, care should be taken so that the terminal is configured for operation at 9600 baud rate, with one stop bit, eight data bits and no parity.

a. 'New' IMU (S.N. 24)

The RS-232 serial output consists of a string of 68 ASCII characters sent *asynchronously* at about 12 strings per second. The string is sent at 9600 baud with eight data bits, one stop bit and no parity. One blank space is sent between each word of sensor data. The data can be sent in decimal, hexadecimal or binary format. The **decimal** output on a terminal looks like:

I +000.0 +00.2 123.3 -0.08 -0.14 -1.53 -00.0 -00.1 -00.2 +176

The contents of the output string are formed as follows:

1. A six character string representing the bank angle starting with a "+" or a "-" and followed by four digits for up to +/- 179.9 degrees.
2. A five character string representing the elevation angle starting with a "+" or a "-" and followed by three digits for up to +/- 89.9 degrees.
3. A five character string representing heading angle from 0-360°.
4. Three five character strings representing body axis (x, y, z) accelerations. The range is +/- 4.99 g's.

5. Three five character strings representing body rates by four digits for +/- 99.9 degrees/second.

6. One three character string representing a 0-5 VDC input.

Two other output formats are available, a hexadecimal ASCII format and a binary format. The **hexadecimal** ASCII format consists of 40 ASCII characters followed by a carriage return and a line feed. This data frame gives 16 bit data of Bank, Elevation, Heading, X Acceleration, Y Acceleration, Z Acceleration, X Angular Rate, Y Angular Rate, Z Angular Rate, and altimeter input, plus a carriage return and a line feed. On a terminal display it will appear as follows:

003B00ADF89E0455DF1902BAF9F6DBA90422FFE4

In the hexadecimal format, each data word consists of 4 bytes. There are 10 words in each message. By adding the carriage return and the line feed at the end, the hexadecimal message total comes to 42 bytes, of which 40 bytes is data. All words are in 2's complement format except for the heading and altimeter data. Instructions on how to manipulate the hexadecimal ouputs are discussed in Appendix B, where the interface software drivers for the IMU are introduced.

The **binary** output mode provides the same information as the hexadecimal ASCII output mode, but in a compact binary format. In this mode, there are 20 words sent, that represent ten 16-bit output channels, followed by a carriage return. The ten words consist of two bytes each, and the data total is again 40 bytes.. The MSB bit is always a one, and the remaining bits contain data, resulting in 14-bit resolution. The user should strip off the high bit. The final byte is the ASCII character for a carriage return. This corresponds to the decimal number 13. On this byte alone, the MSB will not be set. The binary information is transmitted at 35 Hz. On a terminal screen, the binary output appears as a short line of fast changing ASCII characters, that keep overwriting each

other. This is due to the fact that most terminals do not support representation of binary data.

b. 'Old' IMU (S.N. 12)

With this IMU, the data is sent in serial RS-232C format, at a rate of 9600 baud, with eight data bits, one start bit, one stop bit, and no parity bit. All data is hexadecimal ASCII and looks like this:

FFAC02392922080407AFF2FB13AA00A8FFD1

Every data stream consists of nine 4-byte words. At the end of each data stream, a carriage return and a line feed are sent. Thus, the length of the message is 38 bytes, of which 36 are data bytes. The hexadecimal data is in two's complement format. Two banks of data are available. Selection of data banks is described below. Data in the two available data banks, is given in Tables 3.3 and 3.4.

Manipulation of the hexadecimal output is performed as described in Appendix B.

10. RS-232 Input Commands

Depending on the unit, the applicable input commands are:

a. 'New' IMU (S.N. 24)

For this IMU, the RS-232 input commands are provided for the purpose of unit test and installation set-up. The same parameters are used as for the output (9600 baud ASCII). Note that calibration of the unit is done via the RS-232 port. It takes combinations of several keystrokes to recalibration; however, randomly transmitting characters may cause calibration changes. The EEPROM map is retained at Watson Industries.

BANK 1	
x Axis Acceleration - F.S.	+/- 3g's
y Axis Acceleration - F.S.	+/- 3g's
z Axis Acceleration - F.S.	+/- 3g's
x Axis Angular Rate - F.S.	+/- 100°/second
y Axis Angular Rate - F.S.	+/- 100°/second
z Axis Angular Rate - F.S.	+/- 100°/second
Magnetic Heading- North = +7FFFH, East = C000H, South = 0000H, West = +3FFFH	
Bank Pendulum - F.S. = +/- 60°	
Elevation Pendulum - F.S. = +/- 60°	

Table 3.3: Data Bank One

BANK 2	
x Axis Magnetometer - F.S.	2042 mG
y Axis Magnetometer - F.S.	2042 mG
z Axis Magnetometer - F.S.	2042 mG
Initialization Countup Timer: 50 Counts = 1 Second	
Temperature Sensor: FFFFH = +100° C, 0000H = -20° C	
Test Word	
Magnetic Heading	
Bank Pendulum	
Elevation Pendulum	

Table 3.4: Data Bank Two

There are three commands intended for use by the user (others are used at the factory for alignment and calibration).

1. An "R" or "r" will set the outputs (analog and serial) to their Reference Command modes. This will also disable the logic input Reference Command until the next time the unit is powered up.
2. An "I" or "i" will clear the Reference Command mode if it had been set by the serial input.
3. An "!" will reinitialize the unit.

The quotation marks should not be sent [Ref. 16].

There are three output format serial commands: "OD" or "od" as in "output decimal", "OH" or "oh" as in "output hexadecimal" and "OB" or "ob" as in "output binary". These are stored in the unit on EEPROM and are nonvolatile.

The commands "R", "X", "B", "M" display alternate information. The commands "@", "*", "<" and "?", are used by the Watson factory to calibrate the unit and should be used only with the assistance of the factory. The command "+" restores operation, but will not undo calibration changes [Ref. 16].

b. 'Old' IMU (S.N. 12)

The user can transmit commands to this unit to change data being viewed and to exit initialization. (WARNING: Calibration of this unit also is done via the RS-232 port, and randomly transmitting characters may cause calibration changes. The EEPROM map is at Watson Industries). The following characters may be sent on the receive line (see connector pinout, Table 3.1) at any time:

1. An "I" will select Bank One of Data.
2. A "R" will select Bank Two of Data.
3. A "Q" will exit Initialization.
4. A "W" will Re-enter Initialization.

The quotation marks should not be sent [Ref. 16].

11. Error Limiting

To control the effects of error accumulation, a balance of time constant and error limits are used. In addition, centrifugal force corrections also enhance bank and elevation angle quality; however, velocity information is required to utilize this option. The limiting factors that determine error cutout are tabulated in Table 3.5 [Ref. 16].

12. Initialization

The unit enters initialization upon power-up. The complete initialization will last 43 minutes. The length of this initialization can be changed through the RS-232 port. Watson Industries should be consulted, if this time constant needs to be changed [Ref. 16].

Axis of Error Limit	Bank Angle Error	Elevation Angle Error	Centrifugal Force Offset	Acceleration Force Offset
Bank	+/- 11°	None	+/- 45°	None
Elevation	None	+/- 11°	None	+/- 45°
Yaw	None	None	None	None

Table 3.5: Limiting Factor Causing Error Cutout

The initialization is intended to remove bias errors from the accelerometers and the rate sensors. During initialization, the unit should not be moved. The bias initialization on the accelerometers is accomplished using attitude information from the pendulums. This allows initialization even if the unit is not entirely level. The initialization of the rate sensors assume zero rate. The rate sensors have a certain amount of warm-up drift (less than 2°/second) which can last up to 10 minutes. In addition, the sensor bias will change over temperature. A temperature sensor is used to improve bias performance over temperature. A time constant of 80 seconds is used in the initialization of the rate sensors. This allows very accurate short-term information even during warm-up. The user can

enter or leave initialization through the use of commands mentioned above, in the description of the RS-232 input commands [Ref. 16].

E. CHAPTER SUMMARY

Strapdown INS is the navigation system selected for use on the ARCHYTAS UAV. Two strapdown INS suites are available, one analog and one digital. Having described these two INS suites, the next chapter provides a background of the controller design performed for the SAS. The steps taken for integration of the INS with the flight controller are discussed in Chapter V.

IV. CONTROLLER OVERVIEW

The essential avionics component for successful flight of an inherently unstable platform such as the ARCHYTAS is the Stability Augmentation System (SAS). Tasks of the modern aeronautical controls engineer to develop a SAS, include the development of a dynamic aircraft model, verification of its accuracy, design of a control system, and implementation of the controller on a computer prior to flight testing. In earlier days the design procedure was extremely time consuming, and every modification thereafter caused significant delays [Ref. 5]. Fortunately, application software tools such as $MATRIX_X$ with SystemBuild, minimize the time required for the completion of the design, allowing the process to be completed and tested at the block diagram level. In particular, AC100 simulation tools developed by Integrated Systems Incorporated allow for direct conversion of the block diagram to a fully implemented control system. This system can be tested in real-time, in hardware in the loop simulation, while recording any or all of the state variables to verify performance [Ref. 5].

A detailed description of the ARCHYTAS controller design, implementation and hardware-in-the-loop testing, prior to this research, is provided by Moats [Ref. 5]. For comprehension of the concept and operation of the ARCHYTAS SAS, this Chapter contains a generic outline of the aircraft model and controller design. Integration of the flight controller with the INS is described in Chapter V.

A. DESIGN PROCEDURE

The first step in the design process is to create a high fidelity nonlinear model of the aircraft which can be reliably trimmed and linearized throughout the full spectrum of required flight conditions. Such modeling is completed in four stages [Ref. 5]:

- Developing the equations of motion. Sum all of the forces and moments involved and write the equations in terms of states to allow for easy conversion into a block diagram.

- Creating a nonlinear model. Create a block diagram representation of these equations.
- Creating a linear model. Trim the nonlinear model about the desired trim point and then linearize the model to obtain a linear model.
- Validating the model. Use the data from an independent source to validate the model.

The next step is to design a feedback controller. This is typically an iterative procedure beginning with the design requirements. Usually requirements will be placed by the designer in terms of response time and overshoot for the desired controller. For example a heading controller may be required to achieve a ten degree heading change within 30 seconds and have a maximum overshoot of one degree. With the requirements on hand, the control design steps are [Ref. 5]:

- Building a control synthesis model. The designer chooses which sensors to use and creates a synthesis model with the desired command inputs using the linear model.
- Computing the control gain. A cost function is defined by the designer depending on the specified requirements and the method chosen for computing the controller. The control gain is then calculated. The methods for computing this gain include:
  - Linear Quadratic Regulator Theory.
  - H_∞ Theory.
- Check the command and control loop bandwidths: The bandwidth is obtained from Bode plots of the command loops and checked against the specified requirements. The control loop bandwidth is also checked and compared to the bandwidth of the chosen actuators.

The cost function defined above includes some weighting factors. These are used to create the desired controller. If the control and/or command bandwidths are too large

or too small, the designer adjusts these weights and re-computes the control gain. This typically involves many iterations. The designer may also find it necessary to move the zeros placed in the synthesis model and restart the iterations from that point.

The next step in the design process is to test the feedback system. SystemBuild has built-in capabilities for performing these tests. The first test is to close the loop with the linear model and the controller. The resulting closed-loop system is then tested. Next the controller is connected with the nonlinear model and another test is performed.

An important phase of the testing process is the development of an accurate model of the actuators. If the actuators are not modeled well, the software test of the controller may indicate that the controller works as designed, while a hardware-in-the-loop test may show that the feedback system is unstable. This usually happens when the actuators cannot respond fast enough to controller commands.

Once an accurate model of the actuators has been developed, the closed loop software test is repeated with the actuator models in the loop. If the actuator models are accurate and the controller design is correct, there should not be an apparent difference in the performance of the controller.

The designed controller must then be implemented on a platform capable of producing the required control signals and reading the given sensor outputs. Since personal computers, or PCs have become very cost effective, the typical controller implementation is a PC microprocessor with input/output, or I/O cards. The I/O cards available can produce or read analog voltages, Pulse Width Modulated, PWM, signals, and serial communications to name a few. The control algorithm is then programmed using a high level language such as C, and compiled to run on the chosen PC. During programming, careful consideration must be given to initializing and using the I/O cards. Timing is the most critical part of the programmed controller.

The next step is Hardware-In-The-Loop (HITL) simulation, where the feedback system is tested with some of the actual hardware which will be used to control the aircraft. HITL testing is done in one or more stages [Ref. 5].

- Actuators and Sensors: The first stage is to include all of the actuators which receive control signals directly, such as the elevators, rudder, and ailerons. The sensors which measure the results of these actuators, must also be included in order to close the loop. After this initial test, other less critical actuators and sensors may be added. Inertial Navigation System sensors, that provide angles and rates, may be included for proper operation verification, but an extended in-lab test cannot be performed, since actual aircraft motion at these early stages, is not usually allowed.

- Control Inputs and Sensors: The second stage is to include the control input device, such as a joystick for an unmanned aircraft, into the loop along with the sensors required to measure the control inputs. This step has already been done, in the existing form of the ARCHYTAS controller.

The most critical part of any HITL test is calibration of the sensors. The software includes an algebraic conversion of the measured sensor outputs to a signal that can be used directly by the controller. This algebraic conversion requires calibration by determining the correct conversion constants.

The ultimate test of a designed controller is the flight test. Budget considerations demand that the controller works perfectly the first time. The flight test phase is usually performed in three stages [Ref. 5]:

- **Test Stand**: The first flight test is usually done in a laboratory facility on a test stand which allows some degree of movement, while restricting flight so that the vehicle can not be damaged.

- **Tethered Flight**: The next stage is typically a tethered flight. In this manner, an experienced pilot can be standing by with a manual override switch to allow direct manual control of the vehicle in the event of a problem.

- **Actual Flight**: The final and ultimate test of the control design is the autonomous flight test.

The tools currently used at NPS for the SAS design process, are Integrated Systems Incorporated's (ISI) $MATRIX_X$, SystemBuild and AC100 Model C30, which run on a UNIX SUN workstation - 486PC configuration. $MATRIX_X$ and SystemBuild are used to design a plant and controller model in block diagram form, and are loaded and run on the UNIX workstation. AC100C30 hardware is installed on the 486PC, while the appropriate software is loaded and executed on both the UNIX and 486PC machines. Actual I/O connections and hardware control are performed through the 486PC, while software control is executed from the UNIX workstation, via Graphic User Interface (GUI) screens. Once the designer develops a plant model and a controller using $MATRIX_X$ and SystemBuild, AC100 can be used to implement and test the controller on actual hardware with a few pushes of a button. Benefits of using the specific software package are:

- Automation using the AC100 Model C30 dramatically improves the time to first prototype. Valuable time is saved by not having to write code for the control computer and the hardware I/O drivers, neither initially nor every time a change in the configuration occurs. The user only needs a basic understanding of the hardware and it's requirements. All required code is generated automatically by the AC100 software [Ref. 5].

- Improvements to the model or controller can be implemented and tested immediately. For the same reasons as above, any changes made at the block diagram level can be tested immediately with a few mouse clicks [Ref. 5].

- Real-time data acquisition allows for detailed analysis of test results. Since all of the inputs and outputs can be recorded at each time step, the data can be scrutinized thoroughly after a test. This is a tremendous help when trying to find errors created by improper timing [Ref. 5].

B. ARCHYTAS MODEL DEVELOPMENT

The development of the ARCHYTAS model used by the SAS, as described in detail by Moats [Ref. 5], is based on computer modeling of the full non-linear aircraft equations of motion [Ref. 14]. This section begins with an introduction to notation and conventions used throughout the design. Next the equations of motion used in the aircraft modeling are presented. A discussion of the computer modeling of the equations of motion and the development of a SystemBuild model is provided. Finally, testing for the complete ARCHYTAS model is described and analyzed.

1. Notation and Conventions

In order to make this report consistent with the existing research on the ARCHYTAS SAS [Ref. 3, 4, 5], the same notation has to be introduced. Considering Figure 4.1, we define:

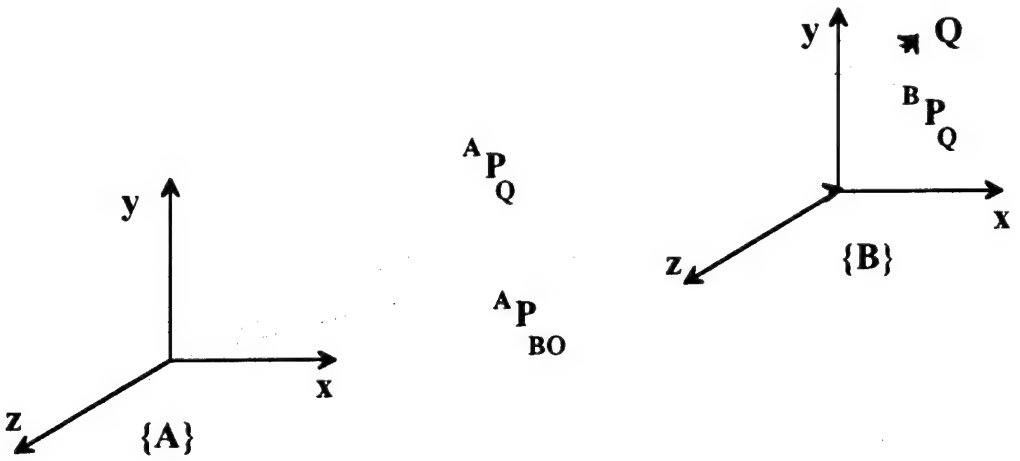


Figure 4.1: Coordinate Systems Relative Position

- {A} represents the coordinate system with basis vectors, x_A, y_A, and z_A
- ${}^{A}P_{Q}$ represents the position of point Q, expressed in{A}.
- ${}^{A}V$ represents the velocity of point Q, measured in {A} and expressed in {B}.

- ${}^{B}({}^{A}V_{Q}$ represents the velocity of point Q, measured in {A}, and expressed in {B}.
- ${}^{A}_{B}$ is a rotation matrix from {B} to {A}, also called a direction cosine matrix. A *free vector* in one coordinate system, is a vector that can be moved parallel to itself without change. As a result, it can be expressed in another coordinate system by using the rotation matrix. For example

$$ {}^{A}({}^{B}V_{Q}) = {}^{A}_{B}R\,({}^{B}V_{Q} \tag{4.1} $$

- ${}^{A}\Omega_{B}$ is the angular velocity of the {B} coordinate system with respect to {A}, and expressed in {A}.
- ${}^{B}({}^{A}\Omega_{B}$ is the angular velocity of {B}, with respect to {A}, and expressed in {B}.

Additional information can be added to the superscripts and the subscripts, i.e., ${}^{A}P_{BO}$ is the position of the origin of {B}, in {A}. For the ARCHYTAS model, the following coordinate systems and assumptions will be used:

- {U} represents the tangent plane coordinate system attached to Earth.
- {B} represents the body fixed coordinate system.
- {W} represents the wind axis coordinate system.
- All sensors are assumed to be located at the c.g., for simplification [Ref. 14].
- The mass of the aircraft remains constant.
- Given a vector v, its derivative with respect to {B} is denoted as $\frac{d}{dt}(v)$, and its derivative with respect to {U} is denoted as $(\dot{v})$.

The {B} coordinate system is a right handed system with X_B defined as the thrust axis. A positive roll rate, p, is clockwise when looking in the positive X direction. The positive direction for Y_B, the pitch axis, is out the right wing. A positive pitch rate, q, is

defined as clockwise in the positive Y direction. The Z_B axis is the yaw axis, and a positive yaw rate, r, is defined as clockwise, looking in the positive Z direction.

The {W} coordinate system is defined with X_W aligned with the wind incident on the aircraft. The angle α is the angle formed by the body x-y plane and the positive X_W axis. The angle β is the angle formed by the body x-z plane and the positive Y_W axis.

The spatial orientation of a rigid body can be defined by the three Euler angles, Φ, Θ, and Ψ, called roll, pitch and yaw.

Further notation simplifications are summarized below [Ref. 5]

- v_Q represents the velocity of an arbitrary point, Q, measured and expressed in {U}.
- v_{BO} represents the velocity of the origin of {B}, measured and expressed in {U}, i.e., ${}^{U}V_{BO} = v_{BO}$
- $\dot{v}_B$ represents the acceleration of {B} with respect to {U}, measured and expressed in {U}.
- ${}^{B}v_Q$ represents the velocity of point Q, measured in {U} and expressed in {B}, i.e., ${}^{B}({}^{U}V_Q) = {}^{B}v_Q$
- ω_B represents the angular velocity of {B}, measured and expressed in {U}, i.e., ${}^{U}\Omega_B = \omega_B$.
- ${}^{B}\omega_B$ represents the angular velocity of {B}, measured in {U}, and expressed in {B}, i.e., ${}^{B}({}^{U}\Omega_B) = {}^{B}\omega_B$

Finally, for better comprehension of the equations of motion to be introduced, the following notation and definitions are introduced

- ${}^{B}\omega_P$ represents the angular velocity of the propeller.
- ${}^{B}F_{GRA}$ represents the force acting on the body, due to gravity. It is

$$ {}^{B}F_{GRAV} = {}^{B}_{U}R\ {}^{U}F_{GRAV} \tag{4.2} $$

with

$$ {}^{U}F_{GRAV} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.3} $$

- ${}^{B}F_{PROP}$ and ${}^{B}N_{PROP}$ represent the force and moment acting on the body, due to propulsion, or thrust. These were measured by B. Stoney [Ref. 1], and are given by

$$ F_{PROP} = 0.0297 \times \delta_{rpm} - 104.7 \tag{4.4} $$

and

$$ N_{PROP} = -0.0542 \times F_{PROP} - 0.9138 \tag{4.5} $$

- $\bar{q} = \frac{1}{2}\rho V^2$ is the dynamic pressure.
- $\bar{S}$ = diag{−S, S, −S, Sb, Sc, Sb is a matrix defined for ease of calculations.
- Control inputs are represented by the vector Δ

$$ \Delta = \begin{bmatrix} \delta_e \\ \delta_r \\ \delta_a \end{bmatrix} \tag{4.6} $$

where $\delta_e, \delta_r,$ and δ_a are the elevator, rudder, and aileron inputs respectively.

- δ_T is the throttle control input.
- Let C_F be the vector of normalized aero forces and moments. Then its partial derivatives are given by

$$ \frac{\partial C_F}{\partial x'} = \begin{bmatrix} C_{L_U} & C_{L_\beta} & C_{L_\alpha} & C_{L_p} & C_{L_q} & C_{L_r} \\ C_{Y_U} & C_{Y_\beta} & C_{Y_\alpha} & C_{Y_p} & C_{Y_q} & C_{Y_r} \\ C_{D_U} & C_{D_\beta} & C_{D_\alpha} & C_{D_p} & C_{D_q} & C_{D_r} \\ C_{l_U} & C_{l_\beta} & C_{l_\alpha} & C_{l_p} & C_{l_q} & C_{l_r} \\ C_{m_U} & C_{m_\beta} & C_{m_\alpha} & C_{m_p} & C_{m_q} & C_{m_r} \\ C_{n_U} & C_{n_\beta} & C_{n_\alpha} & C_{n_p} & C_{n_q} & C_{n_r} \end{bmatrix} \tag{4.7} $$

$$\frac{\partial C_F}{\partial x'} = \begin{bmatrix} C_{L_\beta} & C_{L_{\dot\alpha}} \\ C_{Y_\beta} & C_{Y_{\dot\alpha}} \\ C_{D_\beta} & C_{D_{\dot\alpha}} \\ C_{l_\beta} & C_{l_{\dot\alpha}} \\ C_{m_\beta} & C_{m_{\dot\alpha}} \\ C_{n_\beta} & C_{n_{\dot\alpha}} \end{bmatrix} \tag{4.8}$$

$$\frac{\partial C_F}{\partial \Delta} = \begin{bmatrix} C_{L_{\delta_e}} & C_{L_{\delta_r}} & C_{L_{\delta_a}} \\ C_{Y_{\delta_e}} & C_{Y_{\delta_r}} & C_{Y_{\delta_a}} \\ C_{D_{\delta_e}} & C_{D_{\delta_r}} & C_{D_{\delta_a}} \\ C_{l_{\delta_e}} & C_{l_{\delta_r}} & C_{l_{\delta_a}} \\ C_{m_{\delta_e}} & C_{m_{\delta_r}} & C_{m_{\delta_a}} \\ C_{n_{\delta_e}} & C_{n_{\delta_r}} & C_{n_{\delta_a}} \end{bmatrix} \tag{4.9}$$

- C_{F0} is defined to be the vector of steady state coefficients

$$C_{F0} = \begin{bmatrix} C_{L0} \\ C_{Y0} \\ C_{D0} \\ C_{l0} \\ C_{m0} \\ C_{n0} \end{bmatrix} \tag{4.10}$$

and represents normalized forces and moments in trimmed, balanced flight.

- x' is given by

$$x' = \begin{bmatrix} \frac{u}{V_T} \\ \beta \\ \alpha \\ \frac{pb}{2V_T} \\ \frac{qc}{2V_T} \\ \frac{rb}{2V_T} \end{bmatrix} \tag{4.11}$$

- $\dot{x}'$ is given by

$$\dot{x}' = \begin{bmatrix} \frac{b}{2V_T}\dot{\beta} \\ \frac{c}{2V_T}\dot{\alpha} \end{bmatrix} \tag{4.12}$$

Note that the other derivatives are negligible and therefore are not included.

- The most commonly used notation for the state vector is

$$x = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} \tag{4.13}$$

However, the terms x' and $\dot{x}'$ mentioned above cannot be used directly as states. Therefore, we define

$$M : x \rightarrow x' \tag{4.14}$$

and

$$\dot{M}' : \dot{x} \rightarrow \dot{x}' \tag{4.15}$$

where

$$M = \text{diag}\{\tfrac{1}{V_T}, \tfrac{1}{V_T}, \tfrac{1}{V_T}, \tfrac{b}{2V_T}, \tfrac{c}{2V_T}, \tfrac{b}{2V_T}\} \tag{4.16}$$

and

$$\dot{M}' = \begin{bmatrix} 0 & \frac{c}{2V_T^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{b}{2V_T^2} & 0 & 0 & 0 \end{bmatrix} \tag{4.17}$$

are matrices of appropriate dimensions.

- I_B represents the aircraft body inertia tensor.
- I_P represents the moment of inertia of the propeller.
- I_T is the total inertia tensor, defined as

$$I_T = I_B + I_P \tag{4.18}$$

- ${}^{B}_{W}$ represents the rotation matrix from {W} to {B}, and is a function of α and β. The rotation from {W} to {B} is performed by premultiplying the force or moment vector by the rotation matrix. It is

$${}^{B}_{W}R = \begin{bmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha \\ \sin\beta & \cos\beta & 0 \\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha \end{bmatrix} \tag{4.19}$$

- Let

$${}^{B}_{W}T = \begin{bmatrix} {}^{B}_{W}R & 0 \\ 0 & {}^{B}_{W}R \end{bmatrix} \tag{4.20}$$

and

$$M_I^{-1} = \begin{bmatrix} \frac{I_3}{m} & 0 \\ 0 & {}^{B}I_T^{-1} \end{bmatrix} \tag{4.21}$$

where m is the mass of the aircraft.

- 0 represents the appropriate size matrix with all elements equal to zero.
- I_n represents the identity matrix of dimension n.

2. Equations Of Motion

A detailed derivation of the full non-linear equations of motion for the ARCHYTAS was performed by Moats [Ref. 5, 14]. The final form of these equations in state space form, utilizing the notation previously introduced, is

$$\frac{d}{dt}\begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_{B} \end{bmatrix} = \chi^{-1}\left\{\left[\begin{bmatrix} -{}^{B}\omega_{B}\times & 0 \\ 0 & -{}^{B}I_{T}^{-1}({}^{B}\omega_{B}\times({}^{B}I_{T}\,{}^{B}\omega_{B}+I_{P}\,{}^{B}\omega_{P})) \end{bmatrix}\right.\right.$$

$$+M_{I}^{-1}\,{}^{B}_{W}T\bar{q}\bar{S}\frac{\partial C_{F}}{\partial x'}M\left]\begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_{B} \end{bmatrix} + M_{I}^{-1}\left\{\begin{bmatrix} {}^{B}F_{GRAV} \\ 0 \end{bmatrix}\right.\right.$$

$$\left.\left. + \begin{bmatrix} {}^{B}F_{PROP} \\ {}^{B}N_{PROP} \end{bmatrix}\delta_{T}+{}^{B}_{W}T\bar{q}\bar{S}\left(C_{F_o}+\frac{\partial C_{F}}{\partial \Delta}\Delta\right)\right\} \tag{4.22}$$

$$\dot{P}_{BO} = {}^{U}_{B}R\,{}^{B}v_{BO} \tag{4.23}$$

and

$$\dot{\Lambda} = Q(\Lambda)\,{}^{B}\omega_{B}\,, \tag{4.24}$$

where

$$\Lambda = \begin{bmatrix} \Phi \\ \Theta \\ \Psi \end{bmatrix}, \tag{4.25}$$

$$Q(\Lambda) = \begin{bmatrix} 1 & -\cos\Phi\tan\Psi & \sin\Phi\tan\Psi \\ 0 & \cos\Phi\sec\Psi & -\sin\Phi\sec\Psi \\ 0 & \sin\Phi & \cos\Phi \end{bmatrix} \tag{4.26}$$

and

$$\chi = I_{6} - M_{I}^{-1}\,{}^{B}_{W}T\bar{q}\bar{S}\frac{\partial C_{F}}{\partial \dot{x}'}\dot{M} \tag{4.27}$$

In the above equations P is the position vector of the aircraft, and $Q(\Lambda)$ is the matrix of kinematic differential equations.

3. Computer Modeling

Notice that these nonlinear state-space equations of motion are in a generic format. That is, they could be used to represent the ARCHYTAS or any propeller driven aircraft, provided the correct values are used for the stability and control derivatives, $\frac{\partial C}{\partial x'}$, $\frac{\partial C}{\partial \dot{x}'}$ and $\frac{\partial C}{\partial \Delta}$, as well as the forces and moments of thrust, ${}^{B}F_{PROP}$ and ${}^{B}N_{PROP}$. For this reason, the aircraft plant developed by Moats [Ref. 5], was modeled to accept these values from a generic input file. Thus the same model can be used for different aircraft or flight phases, by simply changing the input data to correspond to the desired new aircraft/phase. Validation of this model, was accomplished by entering the appropriate values and comparing the results to existing data. Data from the ARCHYTAS model and controller, developed and tested by N. Sivashankar [Ref. 3], were compared with the SystemBuild model used in the UNIX-486PC setup, as well as the hardware-in-the-loop testing, as outlined by Moats [Ref. 5]. The results proved the validity of the design.

4. SystemBuild Model

The state derivative equations were implemented on SystemBuild using a user code block. This involved writing a C function which was then linked into the SystemBuild diagram. A detailed explanation of how to implement the nonlinear model with user code blocks is provided by Byerly [Ref. 17]. The SystemBuild ARCHYTAS plant, current at the time this thesis research was initiated, is based on the model developed by Moats [Ref. 5]. A detailed explanation of that model is not repeated here.

Beginning with the basic SystemBuild model, the plant was then modified so that it could be implemented on a digital computer, and tested. Since the AC100 Model C30 can not automatically generate code for a continuous-time system, the model must first be discretized. The goal is to simulate a continuous-time system, using a discrete time model. SystemBuild does this by using a very small time step size, with a continuous type integration algorithm. The continuous model can be discretized in SystemBuild with the 'Transform SuperBlock' option under the 'Build' menu [Ref. 5].

5. ARCHYTAS Model Testing

Finally, the performance of the aircraft model should be verified. This was accomplished by replacing the stability and control values with those of a well known aircraft, in this case the Cessna 172. The nonlinear model can then be trimmed at a given flight condition and the eigenvalues of the resulting linear model can be compared to existing data.

The SystemBuild model for the ARCHYTAS presented here was trimmed for hover conditions and linearized. The resulting state-space matrices were identical to the state-space matrices produced by the SystemBuild model developed by N. Sivashankar [Ref. 3], when trimmed and linearized for hovered flight.

C. CONTROLLER DEVELOPMENT

The development of the ARCHYTAS controller used in the SAS, is presented here. This section outlines the steps taken for the controller synthesis and computer modeling. It also provides the procedure for discretization and testing of the controller model, prior to flight testing.

1. Synthesis and Computer Modeling

The controller can now be designed, based on the valid linear model of the aircraft obtained by linearizing the nonlinear equations above. First, the states or outputs that are available for feedback, and the control inputs to be used by the controller, were determined. For the ARCHYTAS, the measured states are roll rate (p), pitch rate (q), yaw rate (r), pitch angle (θ) and yaw angle (ψ). They are represented in state-space form by the state vector

$$x = \begin{bmatrix} p \\ q \\ r \\ \theta \\ \psi \end{bmatrix} \tag{4.28}$$

The inputs are elevator, rudder, aileron, and rpm (revolutions per minute of the propeller as throttle input)

$$\Delta_{input} = \begin{bmatrix} \delta_e \\ \delta_r \\ \delta_a \\ \delta_{rpm} \end{bmatrix} \tag{4.29}$$

Then, the controller synthesis model was developed. As outlined in detail by Moats [Ref. 5], H_∞ synthesis was used to design the state feedback controller for the ARCHYTAS. This part of the controller design will not be repeated in this report.

2. Discretization and Testing

Next, the controller model must be discretized, in order to be implemented on a digital computer. The SystemBuild implementation of a discrete controller uses a state-space block with the appropriate values as a matrix of gains. The discrete controller for the ARCHYTAS, was developed by Moats [Ref. 5].

Finally, the discretized controller model must be tested, for validity of design and implementation. This includes Closed-Loop testing, performed both with SystemBuild and AC100 Model C30, which should produce identical results. Next, connecting the sensors and actuators in the loop, the controller is tested with Hardware-In-The-Loop, for the ultimate verification before the actual flight tests. Care must be taken, so that all the actuators and sensors are modeled properly. Earlier controller versions and testing are given by Moats [Ref. 5]. The controller at its current form, modified for integration with the IMU and the joystick, is presented in Chapter V and shown in Appendix A.

D. CHAPTER SUMMARY

The SAS design procedure is outlined by the development of a dynamic aircraft model, computer modeling, verification of its accuracy, design of a control system, and implementation of the controller on a computer, prior to flight testing. Modern simulation

tools such as the ISI products used at NPS, allow for direct testing of new projects at the block diagram level, thus minimizing the time required for the design.

V. IMU - CONTROLLER INTEGRATION

Having described the platform, the Inertial Navigation System (INS) sensors and the flight controller, this chapter presents the steps taken for the integration of the Inertial Measurement Unit (IMU) into the ARCHYTAS Stability Augmentation System (SAS). First, the system requirements that have to be satisfied are discussed. Then the modifications to the controller hardware and software dictated by the system requirements are covered. Next, the IMU installation considerations are presented. This chapter also discusses the procedure for interfacing the IMU with AC100C30. The serial driver that was developed for the IMU-C30 implementation is also included. Finally, the chapter provides the complete ARCHYTAS SAS setup, as developed for the integration of the IMU into the flight controller. The Watson IMU is the sensor suite used in the implementation in the ARCHYTAS flight controller. Installation procedures refer only to this digital IMU.

A. SYSTEM REQUIREMENTS

Successful functioning of the ARCHYTAS SAS depends on trouble free interfacing between the individual components of the system. One goal of this research was to efficiently incorporate existing hardware and software in order to achieve proper operation. Specific system requirements are

- Integration of the SAS with the control joystick and the digital IMU.
- Capability for easy selection between operation of the SAS using the ARCHYTAS computer model and using actual measured inputs from the IMU (while the UAV is on the ground). Also, capability for automated use of the IMU measured data during flight.
- Capability to select between flight control with use of actual joystick movements and monitor commands.

- Capability to monitor the model and actual flight parameters on the Interactive Animation (IA) displays with easy selection between operation modes.
- One-time easy pre-flight calibration, using a separate calibration screen.
- User friendly operation and display of commands and flight status with the AC100C30 Graphical User Interface (GUI).

To satisfy these requirements, a number of specific tasks had to be performed. These tasks included

- Modification of existing versions of the ARCHYTAS AC100C30 controller to enable the integration of the control joystick and the IMU.
- Modification of related Interactive Animation displays.
- Installation of the digital IMU on the UAV.
- Development of a serial driver for interface between the AC100C30 controller and the IMU.

The steps undertaken for each task are presented below.

B. CONTROLLER MODIFICATION

To help understand the discussion that follows, a brief introduction of the controller hardware is provided. More details can be found in Chapter VI. The AC100 Model C30 I/O board can carry up to four modules (IP modules) with different functions, such as Analog-to-Digital conversion (IP_HiADC) or Digital-to-Analog conversion (IP_DAC), serial digital input/output (IP_SERIAL), and pulse-width modulated output (IP_PWM) [Ref. 18, 19]. The analog actuator sensors and joystick are connected to an IP_HiADC module, while the digital IMU sensor suite is connected to an IP_SERIAL module. Commands to the vane and throttle servos are currently sent through an IP_PWM module, for operation with a tethered connection. Future plans for an untethered operation via an RF uplink, require the commands to the vane servos to be sent through

the IP_DAC module (see Section G of this Chapter). Operator interface is performed via the AC100C30 Graphical User Interface (GUI) Interactive Animation (IA) screens, which will be covered extensively in later sections.

The SystemBuild project for the ARCHYTAS SAS developed to satisfy the above requirements is called 'flight_test_1'. Its highest level is shown in Figure 5.1 and consists of a SuperBlock for the plant kinematics ('plant_kinematics'), a SuperBlock for the flight control processes ('control_process'), the two necessary SuperBlocks for integration of the IMU ('imu_logic') and the control joystick ('joystk_logic'), and a throttle command conversion block. All the SuperBlocks included in the SAS are presented in Appendix A. The contents and functions of the four major SuperBlocks are explained below.

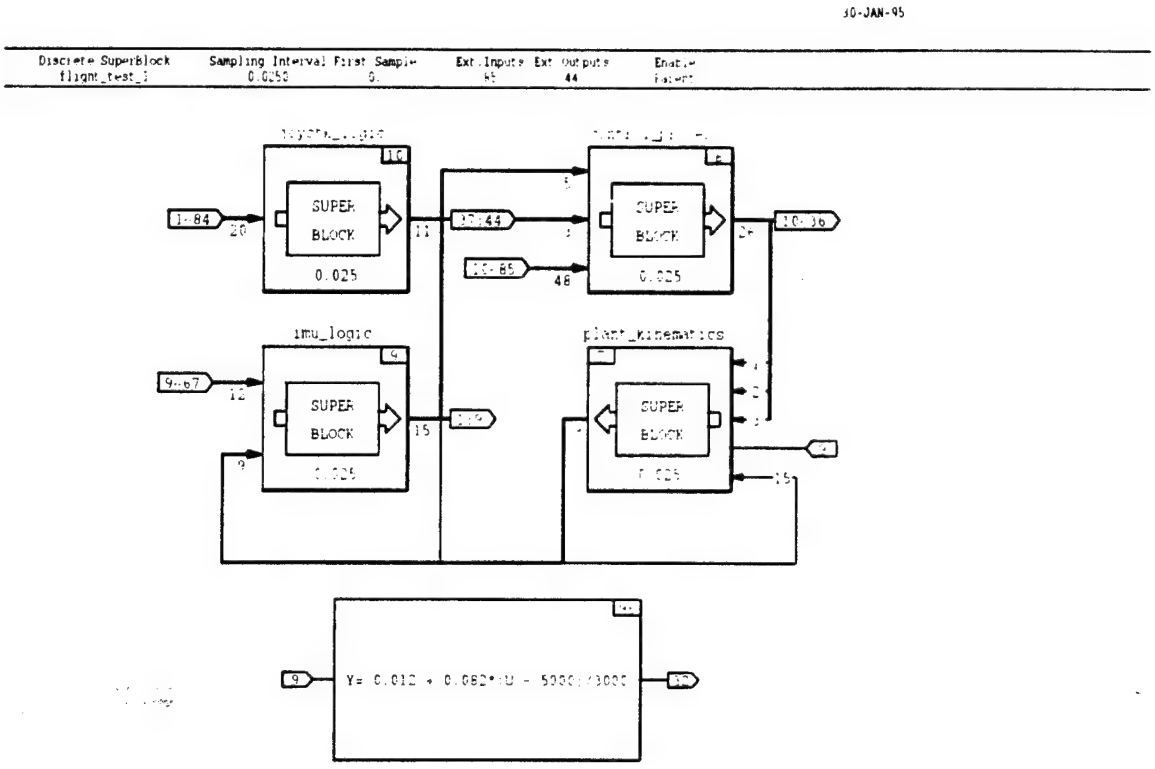


Figure 5.1: ARCHYTAS SAS Highest Level

1. 'plant_kinematics' SuperBlock

This SuperBlock consists of an input block for the constants, a SuperBlock for the aircraft dynamics ('kinematics'), and a SuperBlock for all the integrators ('Integ_sim'). The 'kinematics' SuperBlock is made up of three Blocks representing the angular velocity equations ('ang_velocity_eq'), the linear velocity equations ('lin_velocity_eq'), and the Euler angular rates ('L_dot_eq'). It contains the full non-linear model of ARCHYTAS, without aerodynamic coefficients for the aircraft (Drag forces). Vane aero effects are taken into account. The 'L_dot_eq' SuperBlock implements Equation 4.24 directly. The 'lin_velocity_eq' SuperBlock represents the linear portion (the $\frac{d}{dt}^{B} v_{BO}$ portion) of Equation 4.22. The 'ang_velocity_eq' SuperBlock implements the angular portion (the $\frac{d}{dt}^{B} \omega_{B}$ portion) of Equation 4.22. These values are then fed into an integrator block to determine the states. The 'T_value' block contains equation 4.4 for the engine thrust, while the 'l_m_n_compute' block computes the corresponding moments.

The 'Integ_sim' block contains all the integrators, and is made up of three SuperBlocks. The 'integ_ang_vel' block for the three angular velocities, the 'integ_lin_vel' for the three linear velocities, and the 'int_ang_sim' for the three aircraft angles. All three integrator SuperBlocks contain three identical switches each ('int_off_sw'), that turn off all nine integrators if the IMU is used, so that the model is not used and is "frozen" at the last state.

2. 'control_process' SuperBlock

Analysis and design of the controller followed the guidelines of Chapter IV and are also described by Moats [Ref. 5]. Modifications were implemented on the controller developed by Moats, in order to provide for the IMU and joystick integration. This SuperBlock consists of a block for the discrete controller ('dcont_wind') and a block for the vanes sensors and servos ('vane4x').

The 'dcont_wind' block performs the control of the states and contains three SuperBlocks, the 'input_to_vane_servos', the 'input_to_model' and the 'feedback_laws'. The 'input_to_vane_servos' block is used during the calibration mode, and directs

commands from the IA calibration screen to the vane servos. This is performed either through the PWM module (tethered operation) with use of the 'calibrate' switch, or the DAC module (RF uplink operation, to be used in the future) with use of the 'calibrate_dac' switch. A vane bias value is entered from the IA screen to 'trim' the vanes to zero position. The 'input_to_model' SuperBlock outputs commands to the kinematics of the model. It contains the 'actuators' block, and two switches that select whether the actuators or their models ('act_mdls_in_loop' switch), and/or the hardware ('hw_in_loop' switch) will be in the loop. For HITL testing, the actual actuators (and not the models) are always connected to the controller. The hardware switch logic determines if the model is driven from the controller commands, or the model is driven from the sensed actual servo positions. The 'actuators' block contains four identical actuator model blocks, one for each vane servo (not used). The 'feedback_laws' block is a Delta implementation of a PID controller [Ref. 14]. It contains gain blocks, integrators and differenced sensor and command inputs.

The 'vane4x' SuperBlock performs two functions. It inputs analog signals from the servo sensors through the ADC module, and after processing, it computes commands to each vane servo. It consists of a 'filters' block, four 'vanex_processing' blocks and a vane bias adjuster, that inputs values from the IA display in the same sense as for the 'input_to_vane_servos' block. The 'filters' SuperBlock contains four Butterworth low-pass anti-aliasing filters for the vane sensors, one for each vane. Each of the 'vanex_processing' blocks contains three major groups of blocks. The top group inputs vane commands in degrees from the IA screen, and converts them to DAC voltage (for use with the RF uplink). This branch is put into operation when the 'DAC_output_switch' is used on the IA screen. The middle group inputs vane commands in degrees from the IA screen, and converts them to PWM signals (in use now, with the tether). The bottom group senses vane voltages from the ADC module and converts them to degrees, for use in the calibration procedure. A switch ('rf_pcm_sw1') is used on this branch since it is important

for scaling, to know whether the PWM or DAC module is used for vane commands. A scaling of +/- 30° is used for the DAC, while +/- 100° is used for the PWM.

3. 'imu_logic' SuperBlock

Inputs to this block are the IMU sensor values from the IP_SERIAL module. After processing, outputs are distributed to other SuperBlocks as well as to the IA screen for monitoring by the operator. It consists of the 'imu_in' block, two switches and a decision block. The switches determine whether the IMU data will be shown on the IA display ('imu_model_show'), and/or whether the IMU data or the ARCHYTAS model data will be used in the control process ('imu_model_use'). When the IMU is used, the model state is "frozen" at its last status. Even when the model use is selected, the decision block makes automatic use of the IMU data, once the revolutions per minute (rpm) of the engine exceed a preset number. Currently this number is set to 6499 and is estimated to be the value for which engine thrust is equal to weight. Actual engine tests will better determine this value. The setting of the 'imu_model_use' switch is critical, since use of the IMU is essential in flight at all times, even when the rpm are below 6499. For example, when the aircraft is approaching for landing or is slowing down, the IMU should be used despite a possible low rpm value. The 'imu_in' SuperBlock receives values from the IP_SERIAL module driver, converts the angles and rates to radians, and transforms the values from inertial to body coordinates. An offset is added to 'θ' since because of IMU orientation, when the angle is in fact 90°, it is measured as zero.

4. 'joystk_logic' SuperBlock

Inputs to this block are analog joystick commands from the IP_ADC module. After necessary processing and conversions, outputs are distributed to other parts of the controller, as well as to the IA screen for operator monitoring. It consists of a 'joystk' SuperBlock, a switch, and gains. The 'joystk_logic_sw' switch is set on the IA screen, and determines whether the flight commands will be accepted from the joystick, or the IA monitor inputs. The 'joystk' block contains the 'rpm_command', 'yaw_command', 'pitch_command' and 'rollrate_command' SuperBlocks. All four are identical in concept,

and are used to convert joystick control movements to command voltages usable by the controller.

5. Conversion Block

This block is used to convert monitor entered rpm settings to outgoing PWM commands for the throttle actuator. It is used both in normal operation and in the pre-flight calibration.

C. INTERACTIVE ANIMATION MODIFICATION

System requirements were set for easy selection between different operation modes of the ARCHYTAS SAS, monitoring of model and actual flight parameters, easy pre-flight calibration and user friendly display of commands and flight status. To meet these demands, the already existing Operation and Calibration IA displays had to be altered, according to the controller modifications discussed above. The new IA displays are presented below.

1. Operation Display

Using AC100C30 GUI capabilities a user friendly IA display ('flight_test_1.pic') was developed and is shown in Figure 5.2. Operation of the switches and displays is drawn directly from the SystemBuild description of the ARCHYTAS SAS, and in most cases is self explanatory.

2. Calibration Display

The calibration screen ('calibrate.pic') is shown in Figure 5.3. It is used during calibration of the ARCHYTAS hardware components, to insure proper operation. This display is linked to the operation screen for simultaneous use when the SAS project is invoked.

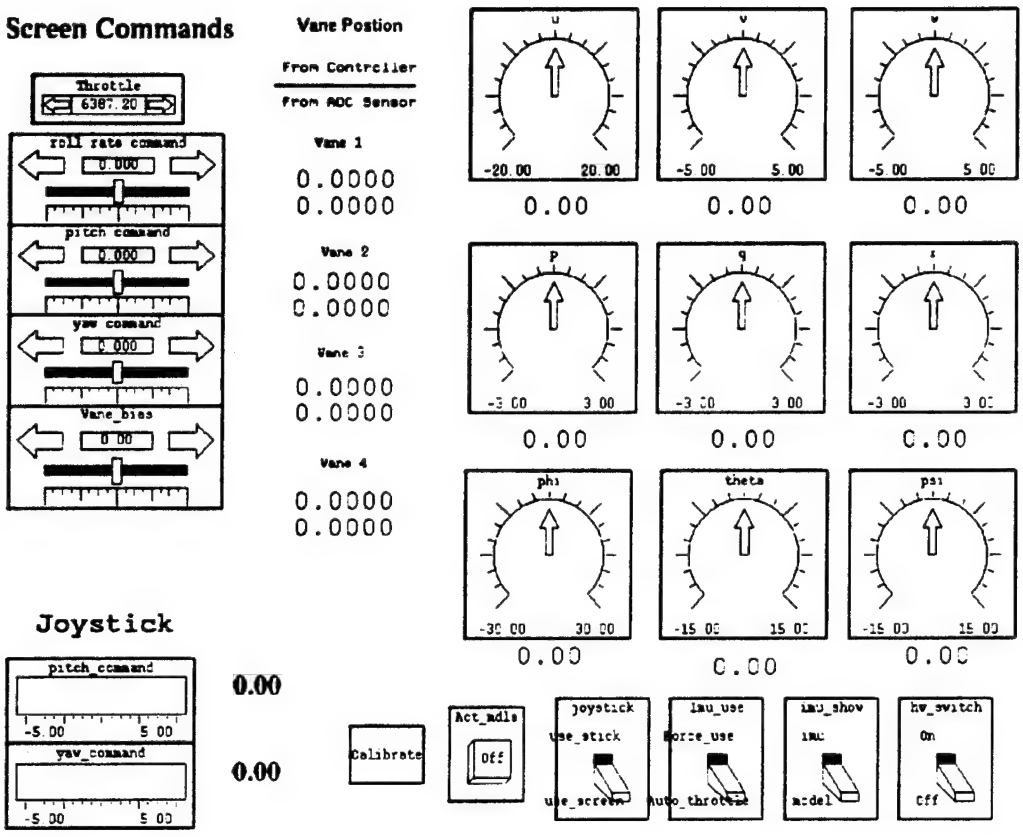


Figure 5.2: Operation IA Display

D. IMU INSTALLATION

The Watson IMU-600D utilizes an earth's magnetic field orientation sensor. A mill aluminum mounting plate is provided for a flat surface mount. The base plate of the transducer is mounted on top of a horizontal surface in the forebody of the ARCHYTAS (Figure 2.1). The Body fixed coordinate system, introduced in Chapter II, is different than the Inertial coordinate system used by the IMU. The IMU-600D is installed with the 9-pin connector toward the forward direction of the vehicle (strut three). Thus, the positive x_I direction is toward strut three, positive y_I direction is toward strut two, and positive z_I direction is aligned with the ARCHYTAS vertical axis, and downward. This is the inertial

orientation assumed in the control process of the flight controller. Both Inertial and Body coordinate systems are shown below in Figure 5.4.

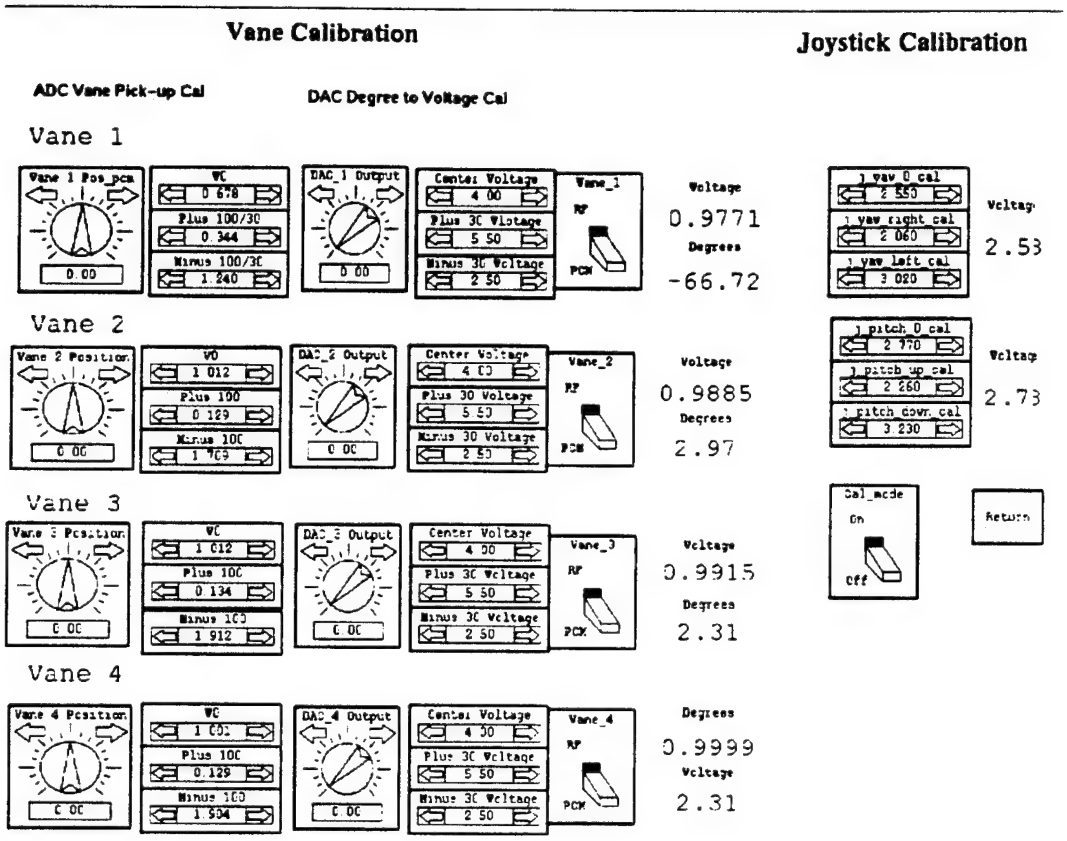


Figure 5.3: Calibration IA Display

Since the SNL analog sensor suite is no longer located in the forebody, there is adequate room for the installation of the IMU-600D. Interfacing of the IMU is accomplished through the 9-pin male connector described in Chapter III. Connection of the IMU-600D with the ARCHYTAS hardware is shown in Figure 5.5. The 24-pin Wiring Harness Tether (see Chapter IV) provides the means for communication between the C30 and the IMU sensor. Power supply to the IMU-600D depends on whether the ARCHYTAS is in flight or not. During in-lab tests power is provided by the wiring

harness through the tether. In flight power is supplied by the ARCHYTAS engine driven alternator.

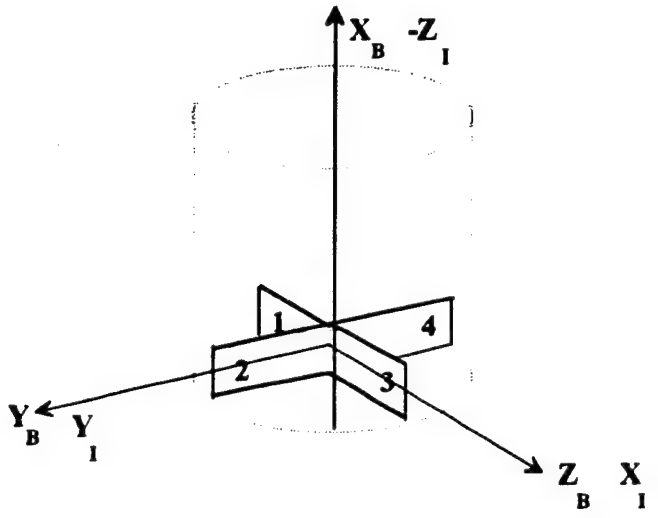


Figure 5.4: ARCHYTAS Body and Inertial Coordinate Systems

E. IMU & C30 INTERFACING REQUIREMENTS

In order to achieve proper communication between the IMU and the AC100 Model C30, data output from the IMU-600D must be received in a format that is fully compatible with the C30 I/O.

The RS-232C serial output format of the Watson IMU-600D is specified in detail in Chapter III. Data from the IMU is received by the C30 through the IP_SERIAL module on the DSP_FLEX board [Ref. 19]. Operation of the IP_SERIAL module is determined by the serial I/O driver included in the C30 software.

The procedure to make external connections to the AC100 Model C30 is provided in detail by Noyes [Ref. 6] and discussed in Appendix B.

F. SERIAL DRIVER FOR IMU & AC100C30 INTERFACE

Following the procedure provided in Appendix B, the serial driver that controls the interfaces of the IMU-600D with the IP_SERIAL module of the AC100 Model C30 was

written in C and includes sections provided by ISI, as well as parts, that were user defined to function properly with the available hardware.

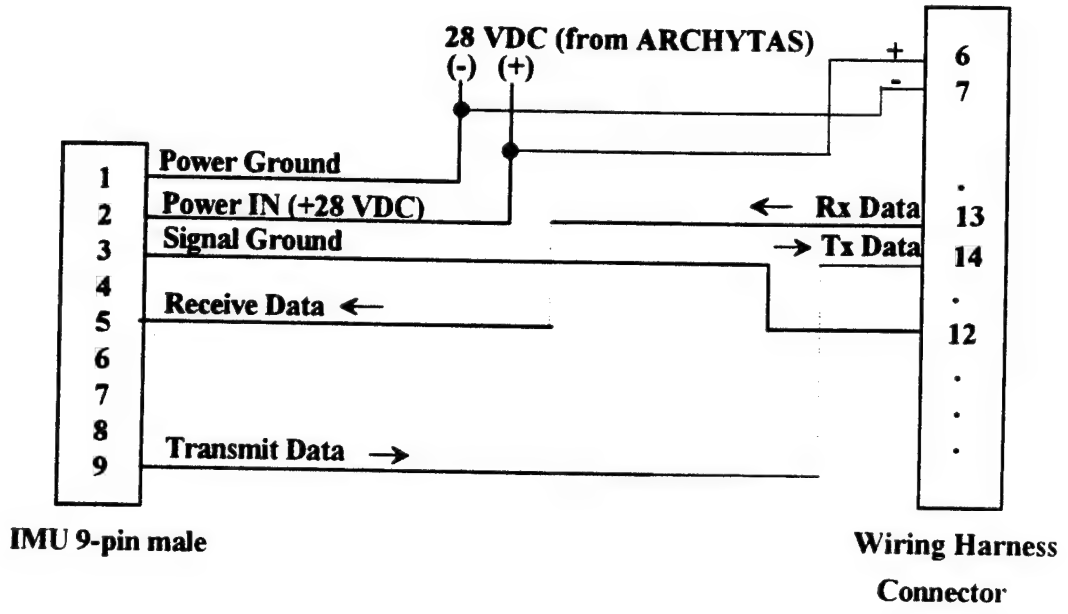


Figure 5.5: IMU - Wiring Harness Tether Connections

ISI supplied the structure for the 'user_ser.c' file. Kuechenmeister added code to the 'user_sample_SERIAL_in' function so that the proper model variables are returned to the simulation. There are three sets of requirements for processing IMU data. The first set is for the 'old' IMU (S.N.12) that is currently in use. This IMU sends out 36 bytes of hexadecimal information at 9600 bps, followed by a carriage return and a linefeed, for a total of 38 bytes. The data is processed in 9 four-byte words by a 'hex_to_integer' conversion routine. The second set is for the hexadecimal output mode of the 'new' IMU (S.N. 24). This IMU sends out 40 bytes of data followed by a carriage return and a linefeed, for a total of 42 bytes. The processing is the same, except that the data is processed into 10 four-byte words. The last set is for the binary output mode of the 'new' IMU. The IMU will send 20 bytes of data, followed by a carriage return. The data is processed into 10 two-byte words, after stripping off the MSB of each byte and applying

two's complement arithmetic to the resulting 14-bit integer. In all cases, the proper scaling factor needs to be applied to each word before it is output to the simulation.

a. IMU-600D/38 and IMU-600D/42 Hexadecimal

The code for the 38-byte messages and 42-byte messages is very similar. In the function 'user_sample_SERIAL_in', the variables need to be sized appropriately. The sizes are shown in the comment header to the function. The 'default_data' variable can be initialized to any values desired. The scale variable must be assigned to values supplied by the vendor. Code in both functions will follow the flow in Figure 5.6. The code for 'user_ser.c' is included in Appendix B.

b. IMU-600D Binary

The code for the binary output mode of the 'new' IMU-600D is similar to the code for the hex output modes. The IMU outputs 10 word messages, so the 'scale', 'last_float', and 'default_data' should be sized at 10. The 'MESSAGE_SIZE' should be sized at 20, and the 'buffer_data' variable should be sized at 25. In the function 'return_model_data', the counter index y should be 10, and the counter index z should range from 0 to 1. The 'hextoi' function has been replaced by a scheme that masks the first bit of each word, then multiplies the MSByte in each word by 2^7 before adding the two 7-bit words together. Then two's complement arithmetic is applied to obtain the proper output for the simulation. The logical flow is shown in Figure 5.6, and the code is included in Appendix B.

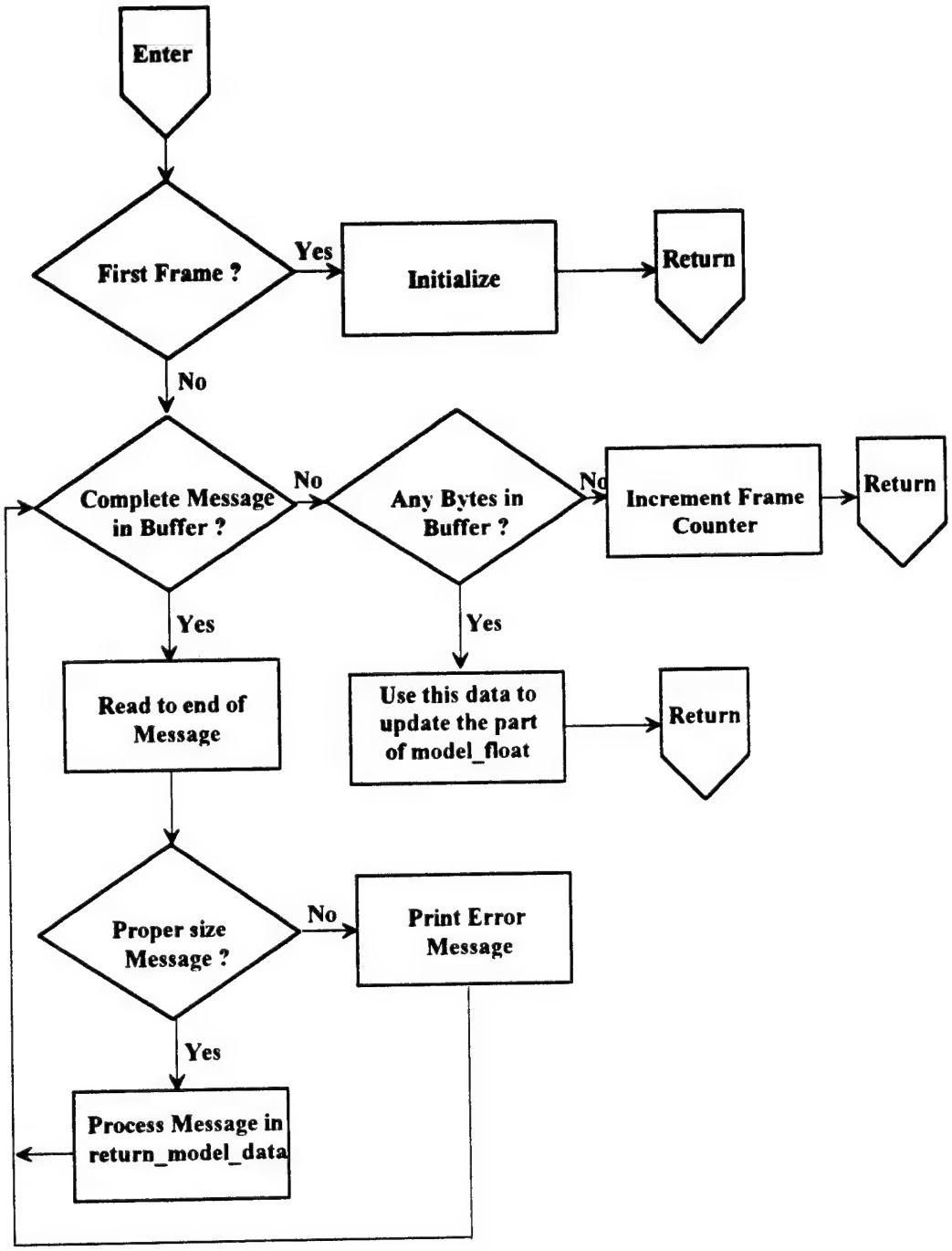


Figure 5.6: Serial Driver Logic Flowchart

G. ARCHYTAS SAS SETUP

The controller for the ARCHYTAS is typically implemented on a microprocessor capable of interfacing with the required hardware. A 486 PC handles communications with the GUI on the UNIX workstation. For HITL testing, the AC100 Model C30, mounted on one of the expansion slots of the PC, executes the control and the plant model software. Eventually the controller will be separated and implemented on the 486 PC processor. In this configuration, the SPARC UNIX workstation acts as the user interface with the C30 controller on the PC. The complete setup for the ARCHYTAS SAS using the AC100 Model C30 for tethered flight is depicted in Figure 5.7.

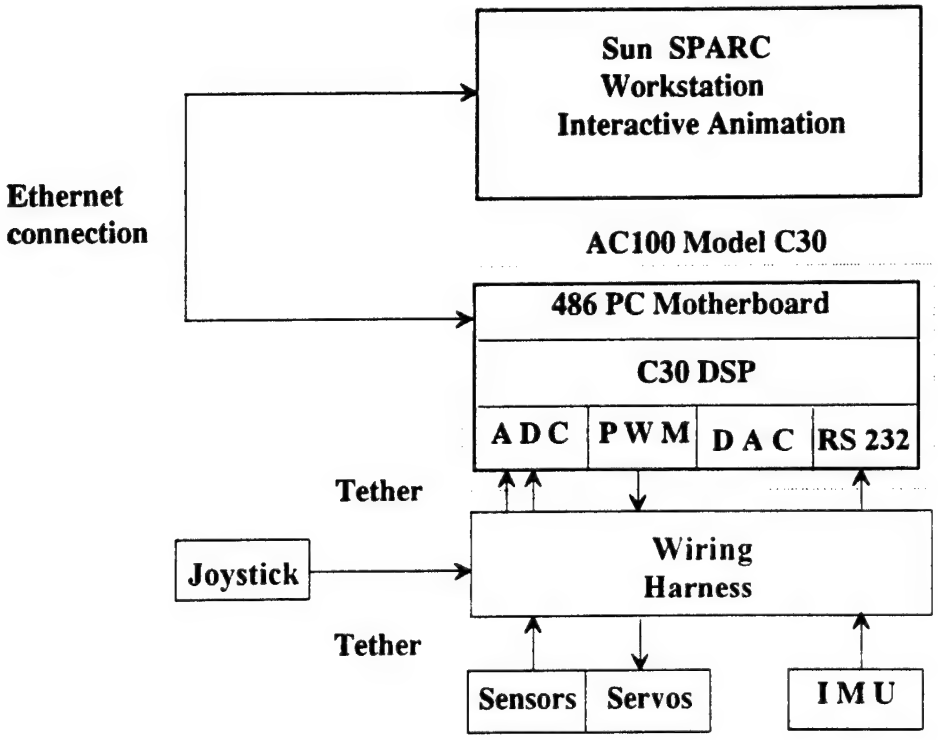


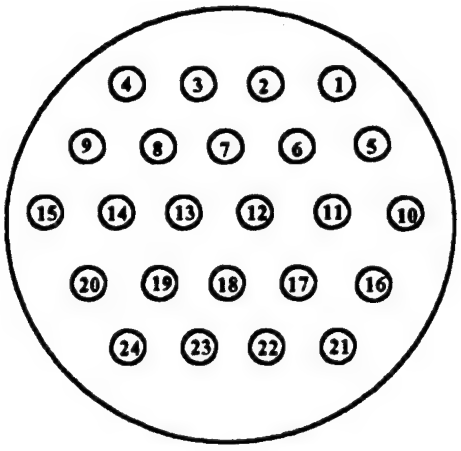
Figure 5.7: ARCHYTAS Hardware Setup, Tethered Flight

In Figure 5.7 we see that connected to the controller are the actuator sensors, the joystick, and the IMU. The analog actuator sensors and joystick are connected to the A/D IP_HiADC module and the digital IMU sensor suite is connected to the RS232 IP_SERIAL module. Commands to the throttle and vane servos are sent through the IP_PWM module. The UNIX SPARC workstation is the user's main interface with the controller. Operator input/output is performed via the AC100C30 GUI screens. The IA picture is updated via the Ethernet connection with the C30, hosted by the 486 PC. The C30 runs the controller and the aircraft model software, and interfaces with all the hardware.

The controller hardware is physically connected to the ARCHYTAS platform via the Wiring Harness. The wiring harness is a signal conditioning box, which was developed so that the first stages of testing - Test Stand and Tethered Flight - could be accomplished, with the control computer connected to the vehicle via a tether. The wiring harness and tether are designed to supply power to all components of the ARCHYTAS, provide all of the control signals, and return all of the sensor outputs.

The tether-harness combination proved to be of invaluable assistance during in-lab tests by providing a rigid hardware connection between the controller and the vehicle, and providing the necessary power to the ARCHYTAS components. Figure 5.8 depicts the current connector end of the tether. The pin diagrams for each of the C30 IP modules are included in the ISI AC100 manuals [Ref. 19]. The complete wiring diagram for the ARCHYTAS HITL test on the C30 is shown in Figure 5.9.

The wiring harness box consists of a PC shell containing screw terminal blocks and a PC power module, as well as a 24 Volt power supply. Connections with the AC100C30 hardware are made possible through two 37-pin ports located on the back of the wiring harness box. Looking from the back of the PC shell, the left 37-pin connector (Port A) is shown in Figure 5.10. The right 37-pin connector (Port B) is shown in Figure 5.11.



Pin Number	Content	Pin Number	Content
1	Vane #1 signal	13	IMU Rx
2	Vane #2 signal	14	IMU Tx
3	Vane #3 signal	15	5V Return
4	Vane #4 signal	16	Vane #1 center
5	Throttle signal	17	Vane #2 center
6	+24 V Power	18	Vane #3 center
7	24 V Return	19	Vane #4 center
8	Kill switch	20	Throttle high
9	Kill switch	21	Vane #1 high
10	Throttle center	22	Vane #2 high
11	+5 V Power	23	Vane #3 high
12	IMU Ground	24	Vane #4 high

Figure 5.8: Connector End of Wiring Harness Tether

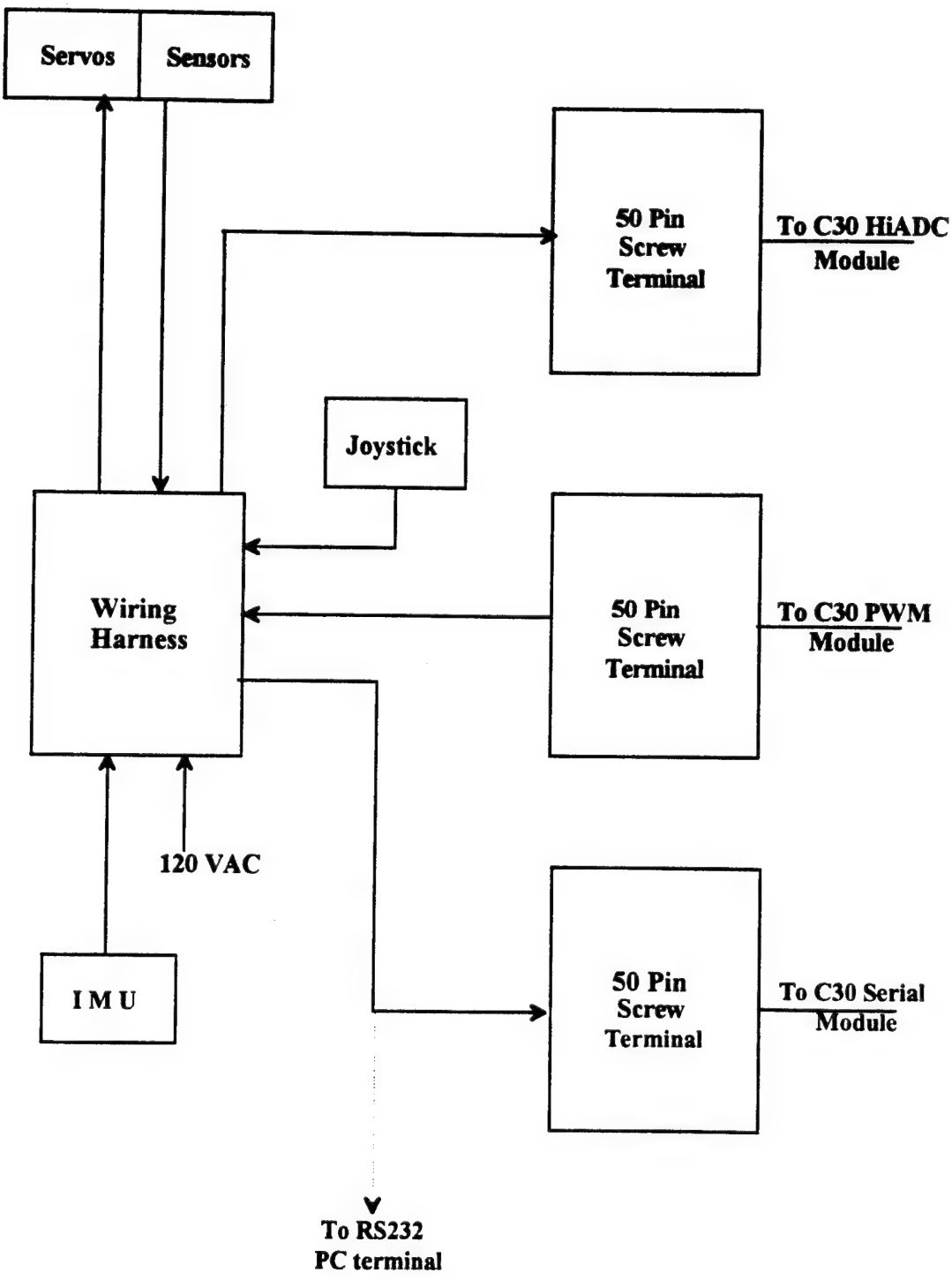


Figure 5.9: AC100 Model C30 HITL Test Wiring Diagram

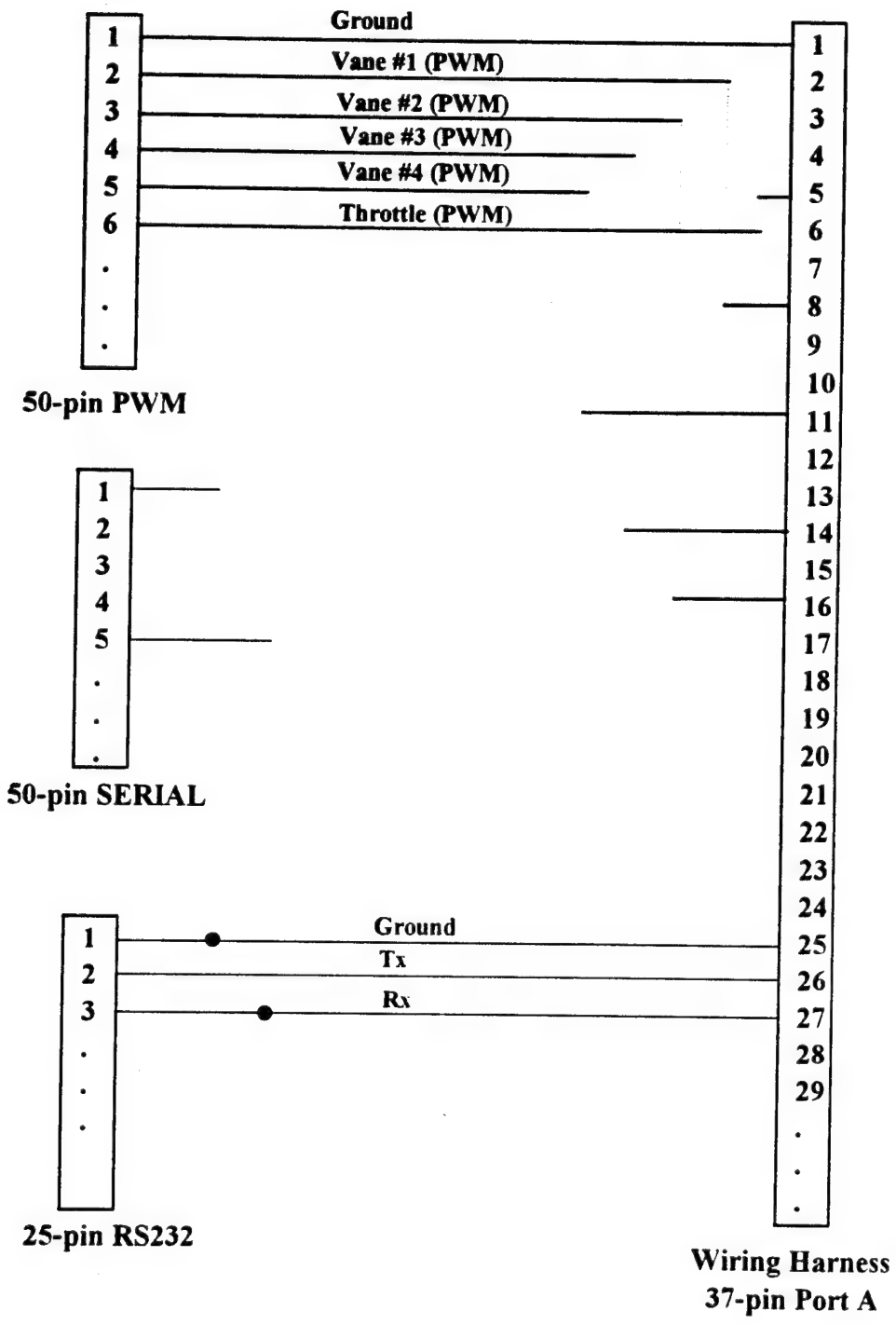


Figure 5.10: Wiring Harness - C30 I/O Connections (Port A)

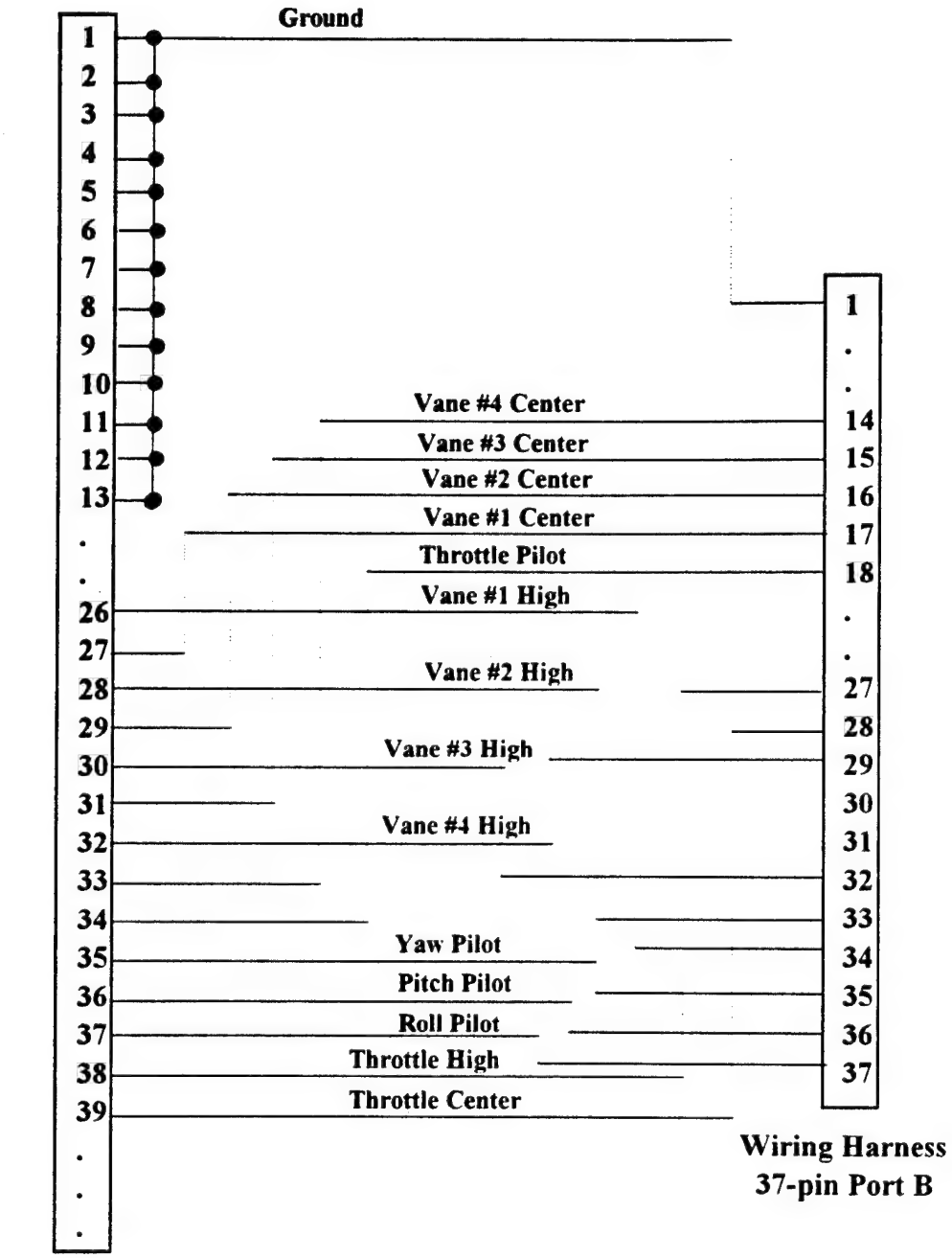


Figure 5.11: Wiring Harness - C30 I/O Connections (Port B)

H. CHAPTER SUMMARY

System requirements for successful integration of the IMU with the SAS lead to modifications of existing versions of the flight controller and related IA displays. Furthermore, the requirements dictated installation of the digital IMU on the UAV and modification of the C30 serial drivers, for proper interface between the IMU and the AC100C30. These tasks lead to the final form of the ARCHYTAS SAS setup, which has been discussed. Flow charts and wiring diagrams have been presented to complete the description of the IMU integration.

VI. SYSTEM EVALUATION AND RESULTS

This Chapter presents the tests performed to evaluate the integration of the IMU with the ARCHYTAS flight controller, in terms of software design and implementation, as well as vehicle and ground control hardware design testing. Evaluation is accomplished in three steps; closed-loop testing, hardware-in-the-loop testing and flight testing. Results are discussed for each step of the validation. Conclusions drawn from the results, and recommendations for improving the system based on the experience gained, are presented in the final chapter of this report.

A. CLOSED-LOOP TESTING

The objective of closed-loop testing was to validate the SAS development by comparing the controller performance with the design requirements. This test was performed in three stages. First the feedback system was tested with the linear model and the controller. Then the linear plant was replaced by the nonlinear model. Finally, after an accurate model had been developed for the actuators, the closed loop system was evaluated with the actuator models in the loop. If the modeling is correct, results in the last two stages should be very similar.

Special attention should be paid to actuator and sensor modeling. The development of an accurate model for actuators and sensors is essential to the success of the testing process. An incorrect modeling might lead to inaccurate results during the software testing of the controller. In that sense, a controller that functions as designed during a software test, might prove to be unstable during a later HITL feedback system test. This is usually due to inability of the actuators to respond to controller commands as fast as the modeled actuators, or possibly an unacceptable quality of information provided by the sensors. [Ref. 5]

Proper modeling of the actuators requires the determination of the corresponding transfer functions. Identification based on the step and the frequency response of a system can be used to determine the actuator models. Furthermore, accurate sensor data

processing requires utilizing filters for noise cancellation, elimination of under-sampling and anti-aliasing.

The ARCHYTAS controller uses a fourth order system to model each servo actuator, and a discrete third order Butterworth low-pass filter for each sensor. A detailed development of the actuator models and outline of the sensor design is provided by Moats [Ref. 5].

1. Setup

For closed-loop software testing, the complete model and controller are connected as shown in Figure 6.1. Here P represents the plant model, K represents the controller, y represents the actual plant output, and u_c is the control input created by the controller. The first test of the feedback system, is to close the loop with the linear system and the controller. Next, the nonlinear aircraft model is connected with the controller and tested. The Closed-Loop testing was conducted using SystemBuild, as well as on the AC100C30. Results for these are discussed below.

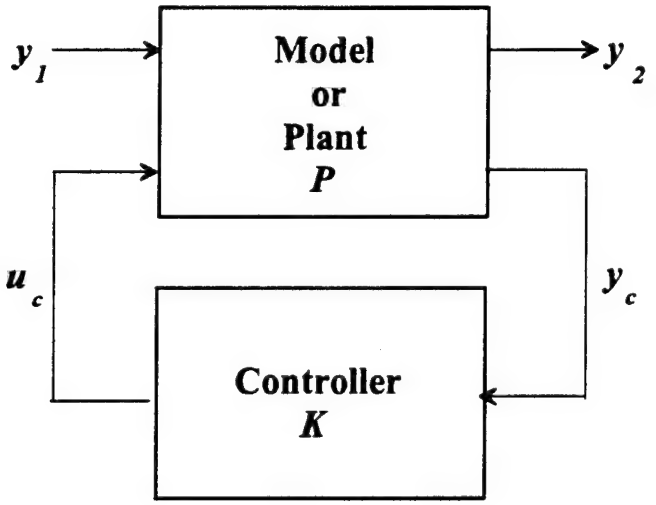


Figure 6.1: Closed-Loop Block Diagram

2. SystemBuild Testing Results

A discrete time controller can be tested with the continuous time aircraft model if the outputs of the discrete controller are sent through a zero-order hold, before being input to the continuous model. This step is done automatically by SystemBuild, when discrete and continuous SuperBlocks are connected within the same block diagram.

To conduct SystemBuild testing the user needs to define a time vector. For a 40 Hz controller, since $\frac{1}{40} = 0.0025$, the time vector, might be:[Ref. 5]

$$t = 0 : 0.0025 : 30; \tag{6.1}$$

This produces a vector of 1201 elements, starting at zero ending at 30 seconds, with spacing of 0.0025 seconds. The user can define an input vector in MATRIX_x or connect a signal generator block inside the SystemBuild model. To enter the appropriate values, 'Analyze SuperBlock' should be selected from the 'Build' menu. Entering 'sim' at the MATRIX_x prompt, begins the test and generates a matrix of appropriate values. This output matrix can be analyzed if broken into vectors and observed using the 'plot' command [Ref. 5]. SystemBuild testing and results that prove the validity of the design for the ARCHYTAS SAS, are provided by Sivashankar [Ref. 3]. Since the basic controller logic has not been altered in this research, this test was not performed again.

3. AC100 Model C30 Testing Results

The discrete model can be tested on the AC100 Model C30 by following the procedures outlined by Moats [Ref. 5] without connecting any of the hardware. The closed-loop connections are left in the model and the desired outputs are selected for the IA display. The test results were identical to those found using SystemBuild. This is to be expected since the C30 processor is using exactly the same closed-loop system as SystemBuild.

B. HARDWARE-IN-THE-LOOP TESTING

The final validation of a controller, prior to an actual flight test, is usually the hardware-in-the-loop (HITL) test of the controller. HITL is the most critical phase of testing, and it involves the actual control processor, actuators, and some or all of the actual sensors. In the case of the ARCHYTAS, the only sensors that can be used for a meaningful HITL in-lab test, are the servo-motor position sensors. The IMU can be included in the lab, but would not produce usable data, since no aircraft motion is possible. Motion of the IMU alone, with the sensor in the loop, can verify the correctness of the hardware connections. Also, observation of control surfaces' tendencies in response to IMU movement, can assist in evaluating the validity of the controller logic. However, the IMU was not initially included in the HITL test, and the sensor was modeled along with the aircraft [Ref. 5]. In later versions of the ARCHYTAS controller, the control joystick and the actual IMU were included in the HITL test, in order to approach as much as possible the final hardware connections of the controller, prior to flight testing.

As stated in Chapter V, the ARCHYTAS is set up to operate connected with a tether to the controller hardware. Future development will lead to completely unattached flight, with RF up and down links, instead of the tether. The procedure described in this section refers to HITL testing for the tethered setup. Development of an RF downlink for the IMU data and HITL testing for the untethered setup is covered in Chapter VII.

1. Setup

To conduct a HITL test using the AC100 Model C30 with SystemBuild, the necessary software tools must be developed and configured. This step is performed on a UNIX workstation, and is dictated by the status of the hardware, and the anticipated HITL results. The AC100 Graphical User Interface (GUI) is the workstation users' link to all the software tools required for modeling and testing.

Once the user determines which outputs to display and which inputs are desired for interaction with the running model and the controller, the next step is to build or modify the Interactive Animation (IA) display, that will be used during the HITL test.

The Hardware Connection Editor (HCE) is the tool used to make connections to hardware and also to the IA picture [Ref. 6, 18]. Here the user defines and manages inputs and outputs to the model, as well as individual hardware modules. There are four slots, or IP slots, available on the AC100 DSP_FLEX carrier board, that can host hardware (IP) modules. Available IP modules are [Ref. 6]:

- IP_SERIAL: For serial input and/or output connections.
- IP_HiADC: Used for input only. Any analog voltage signal can be sampled and used in a digital format.
- IP_DAC: Digital signals are converted to their analog equivalents.
- IP_PWM (actual name IP_68332): Used in many modes for both input and output. In the ARCHYTAS controller, this module is used to generate Pulse Width Modulated signals, by determining the desirable duty cycle.

Note that every IP module can be mounted on any of the four available IP slots, provided the host slot is configured for the specific module operation. Slot configuration is achieved by using the corresponding switch (switch A for slot #1, switch B for slot #2, etc.) [Ref. 19]. For proper operation, when changes are made to the HCE caution must be taken in addressing the modules on the C30 DSP_FLEX board. Since there are currently two C30 carrier 486 PCs available at the UAV lab ('ac100' and 'america'), care must be taken to make sure that the HCE setup, corresponds to the actual configuration that exists on the C30 used each time. Next, all the applicable hardware must be physically connected to the control computer, which is the C30 Digital Signal Processor, located in the PC. The complete setup for the HITL test using the AC100 Model C30, is shown in Figure 6.2.

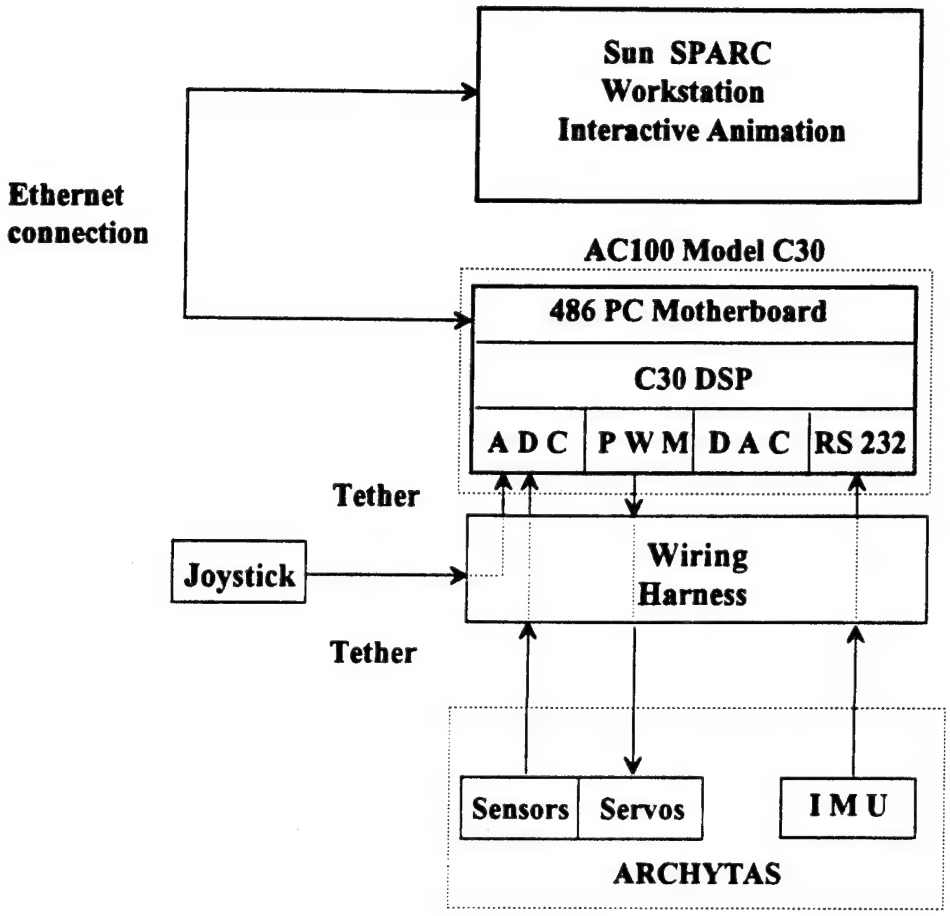


Figure 6.2: ARCHYTAS Hardware-In-The-Loop Setup, Tethered Flight

As shown in Figure 6.2 the analog actuator sensors are connected to the A/D IP_HiADC module, and the digital IMU sensor suite is connected to the RS232 IP_SERIAL module. Commands to the servos are sent through the IP_PWM module, while the joystick is connected to the IP_HiADC module also. The IMU sensors can be either digital or analog, depending on the Inertial Navigation suite selected for use. In the analog case, the IMU would have been connected to an A/D Hi_ADC module, needed for converting the analog signals to the digital form required by the controller. Since the digital IMU is currently the sensor suite of choice, for the rest of this report, it will be assumed that the ARCHYTAS flight controller is configured with a digital IMU.

Before the actuator sensors can be used reliably by the controller, they must be first calibrated. Due to small changes in the operating voltages of the power supply, which affect the analog nature of the sensors, calibration is required each time the controller is started. For a HITL test, a separate C code program has been implemented by Moats, and is run to calibrate the actuator sensors [Ref. 5]. Each time the controller is started, the I/O devices must be initialized. This results in small changes in the measured voltages on the analog to digital I/O device, from one initialization to the next. For this reason a 'chained' IA picture is used, so that the I/O would not have to be re-initialized after calibration. This is the calibration screen shown in Figure 5.3 ('calibration.pic'). The procedure for calibration differs depending on the configuration been tested.

For the tethered case, calibration involves measuring the sensed voltage for vane positions of +/-100 deg, and zero degrees. The model utilizes these measured voltages in an equation used to convert the measured sensor voltage, into the correct vane position in degrees. Commands are then sent through the IP_PWM module. Details of the conversion equation and the calibration procedure, are provided by Moats [Ref. 5]. Once the calibration routine is complete, the user can return to normal operation of the controller. In an attempt to make the procedure straight forward, the calibration screen has been grouped and labeled according to the function performed by each switch and dial.

In the case the analog IMU is employed by the controller, calibration of the sensor suite is required prior to each operation, for the same reasons that applied to the actuator sensors. In the current form of the controller, use of a digital solid-state IMU implies that such calibration is not required, provided the IMU is calibrated by the manufacturer and is in good functioning order. Small variations in the operating voltages of the power supply, do not affect the digital output of the IMU sensors, thus acceptable operation is expected every time the controller is initiated.

2. HITL Results

Once the HITL setup is complete, the test can be accomplished with a few clicks of the 'mouse'. When the HITL test was conducted the ARCHYTAS SAS was found to

have stable response to operator commands. It satisfied design requirements for command and control loop bandwidths and step response, while it demonstrated robustness during operation. The results were the same with HITL tests performed in previous research [Ref. 5] and produced significant insight concerning the validity of the ARCHYTAS controller. Modifications were necessary, which led to the form of the flight controller currently in use. Complete instructions on how to prepare and conduct a HITL test, are covered extensively by Moats [Ref. 5].

C. FLIGHT TEST

The ultimate evaluation of a designed SAS is flight testing. The HITL testing before actual flight assists so that the newly designed or modified SAS is almost ready for first time flight testing. However, time limitations did not allow flight testing to be performed during the course of this research. For this reason, the flight test stages are only mentioned here and are discussed extensivelly in Chapter VIII, as recommendations for future development.

For the ARCHYTAS, the flight test phase is intended to be performed in three stages:

- Test Stand
- Tethered Flight
- Actual Flight

Conducting flight tests outside the lab, requires mobility while still providing all the necessary elements of the SAS. For this reason, a UNIX workstation was configured for standalone operation, along with a portable 486PC on an Ethernet link. These components as well as the wiring harness and the joystick, need to be moved to the flight test site. The equipment required for (tethered) flight testing is shown in Figure 6.3, while the procedure for moving these components outside the lab, is described in Appendix C.

Prior to flight testing, the ARCHYTAS would first be placed on the thrust stand to allow testing of the engine performance, power curves and consumption characteristics.

The new hardware implementation of the latest flight control system can also be evaluated. Any engine calibration that is necessary can be done at this point. Cibula on a concurrent thesis has conducted engine runs on the ARCHYTAS [Ref. 20]. During those tests, the intended aircraft operator has become familiar controlling the throttle with the joystick.

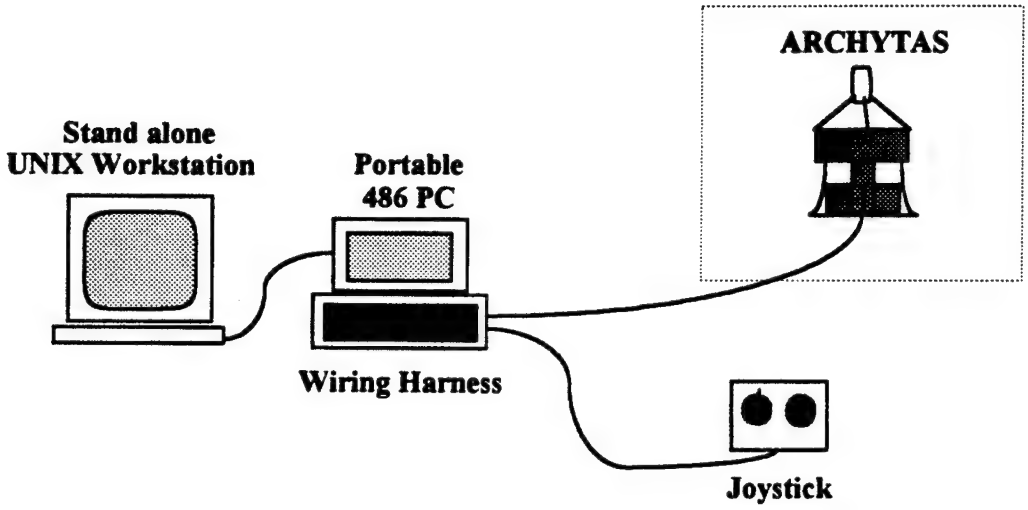


Figure 6.3: Flight Test Setup

Upon completion of those tests, the UAV group intended to install the airborne components of the SAS, and proceed to flight testing. Unfortunately, extensive vibrations during engine operation, lead to numerous electrical problems, as connectors and wires either became loose or completely broken. Replacement of connectors and reinforcement or rewiring of the hardware lead to delay of the first flight test stage. Nevertheless, the setup and procedure for each of the three stages was determined well in advance and is presented in Chapter VIII.

D. CHAPTER SUMMARY

Evaluation of the IMU-Controller integration is performed by closed-loop testing of the controller, HITL testing of the complete setup, and ultimately by the three stages of flight testing. The validity of the design and implementation was verified by meeting design requirements, and by obtaining the same results as tests performed during previous research studies. The ARCHYTAS SAS proved to have stable response to operator commands and exhibited robustness during operation. It satisfied command and control loop bandwidths as well as step response requirements.

VII. DATA LINK DEVELOPMENT

Previous and concurrent researches of the NPS UAV project group converge to totally unattached autonomous flight of the ARCHYTAS aircraft. Eventually, after the tethered flight testing is completed, the tether that connects the ground controller with the UAV will have to be replaced by wireless up and down data links. This chapter discusses the development of a radio frequency (RF) down link, for transmission of IMU data from the UAV to the ground based AC100C30 controller. It presents the specific datalink requirements and continues with a reference to prior research performed on the subject. A commercial off-the-shelf (COTS) RF link is selected to meet the design requirements. This chapter finally includes the implementation and testing of the RF datalink, connected with the ARCHYTAS SAS hardware in the loop.

A. OVERVIEW

During unattached flight the tether will be replaced by wireless radio frequency (RF) data links, that perform the same basic functions as the tether. For proper control of the ARCHYTAS in flight, IMU data has to be transmitted from the UAV to the ground, while servo commands have to be sent from the controller to the aircraft actuators. The IMU data is critical to the control procedure. The serial format of the output as well as the data throughput is of great significance and should not be altered. Furthermore, once the IMU is initiated and configured, its operation does not require user input commands. It was thus determined that a dedicated one-way communication link, a *downlink*, with specific operation characteristics should be employed to provide the IMU output to the C30 controller. On the other hand, servo commands do not impose so strict requirements on link data processing and throughput. Therefore they can be sent on a separate data communication channel, an *uplink*.

A possible wireless configuration is depicted in Figure 7.1. Power in this mode will be supplied by the engine alternator. The wiring harness will be modified, to interface with the uplink transmitter and the downlink receiver. The transmitter of the uplink is

connected to the C30 controller, while the receiver is onboard the ARCHYTAS and is connected to the servos. On the other hand, the downlink transmitter is connected to the IMU installed onboard the UAV, while the receiver is connected to the C30 controller.

The design and development of the RF downlink is the scope of this Chapter. The RF uplink will be based on COTS Futaba model airplane radio components and is not covered in this research. Furthermore, future employment of GPS will require continuous exchange of data between the UAV and the ground station, on a two-way communication link. The development of such a Full Duplex link is in progress at NPS.

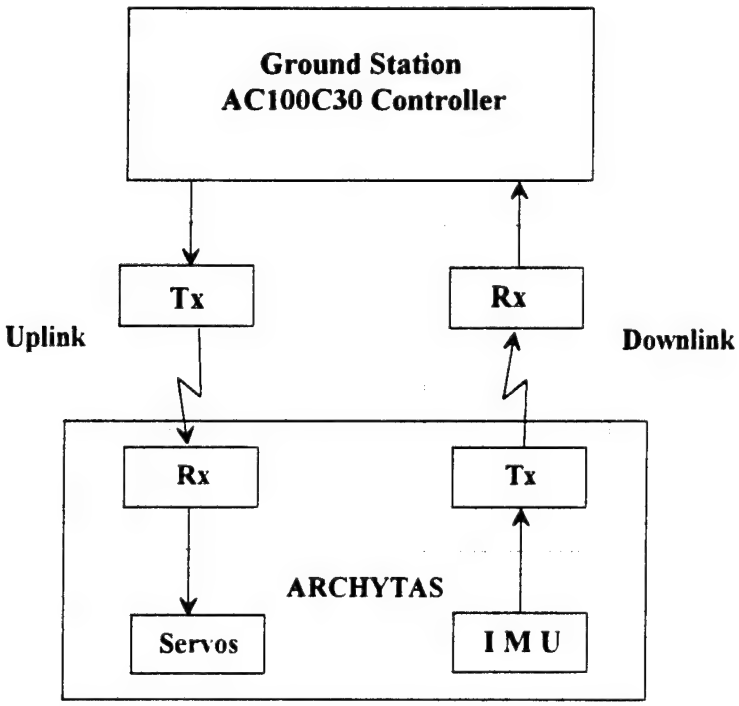


Figure 7.1: Untethered Mode Configuration

B. REQUIREMENTS

As explained in Chapter V, trouble free integration of the flight controller with the INS depends on the successful interfacing of the controller components with the IMU sensor. Having succeeded in developing and testing a properly functioning SAS, any further improvements should attempt to minimize possible changes to the existing setup

structure. In this way, complications and malfunctions with the associated need for troubleshooting can be avoided. From that viewpoint, the wire tether should be replaced with an RF data link that meets the following criteria:

- Cost limitations. The price should not exceed $3,000.
- Weight limitations. The airborne unit should not weigh more than 5 lbs.
- Power limitations. Power requirements should be compatible with the ARCHYTAS alternator.
- Size limitations. The airborne unit should fit in the ARCHYTAS forebody.
- Standard RS-232 serial interface.
- Trouble-free interface with the IMU and the AC100C30 hardware and software. Data exchange at 9600 bps with 8 data bits, 1 stop bit and no parity.
- Ability to transmit/receive beyond the line of sight.
- Frequency agility.
- Hardware reliability.
- Adequate data throughput. The radio link should not add any overhead data to the IMU output, while it should have at least a data throughput of 9600 bps.

The development of an appropriate down link for the IMU was based on finding a COTS system that would most closely meet the above requirements.

C. PREVIOUS RESEARCH

Two different links had already been developed for the ARCHYTAS UAV. Both were commercial off-the-shelf (COTS) products [Ref. 7]. The first datalink solution was a X.25 packet radio terminal node controller (TNC) connected to a 19.2 Kbps modem in combination with a UHF wide-band transceiver, developed by Reichert [Ref. 8]. This was a communication system that met or exceeded criteria set at earlier stages of the UAV

project. At that time a PC based controller was intended to perform flight control of the ARCHYTAS, while another ground based PC served as the user interface.

The second datalink solution was a direct-sequence spread spectrum UHF datalink as developed by Bess [Ref. 9]. This system also used a modified X.25 packet radio protocol with a top transfer speed of 19.2 Kbps. Spread spectrum transmission has the added advantages of being less susceptible to jamming signals and may be operated without an FCC license. It had programmable length packets and performed its own error correction and flow control.

Currently flight control is performed by the ground based AC100C30 controller, while sensors are installed on the UAV. In order to link the IMU with the C30, only transparent datalink operation is allowed, i.e., the data applied to the transmitting component is output by the receiving component without alteration. Neither of the above mentioned systems can be used in the ARCHYTAS SAS setup because of their packet radio operation mode, thus a new system was required.

D. DEVELOPMENT

The aforementioned requirements call for a cost-effective, compact, robust, reliable and agile RF datalink that would interface with the RS-232 serial IMU output and C30 input. It has to be specifically capable of fully transparent operation at 9600 baud per second (bps) minimum, with one stop bit, eight data bits and no parity. After careful survey of available COTS systems, the model SLQ-96 radio link manufactured by Repco, Inc., was found to meet or exceed all of the design criteria. A demonstration system was provided by Repco for evaluation of the data link's capabilities. General information and configuration of the SLQ-96 is discussed below, while technical specifications and detailed operation instructions are included in Appendix D.

1. SLQ-96 General Information

a. Description

The model SLQ-96 radio modem provides high-speed data link between

computers or data terminals operating at data rates of 4800 or 9600 bits per second (bps). The SLQ-96 features a utility enclosure, operates on +12.5 VDC, and if needed, a duplexer must be mounted externally. The modem utilizes separate Repco RDL transmitter and receiver modules and incorporates the Datawave MM9612 modem hybrid. The MM9612 is a single module capable of performing all the functions of a 4800-9600 bps modem and is located on the modem interface board (MIF). A separate processor board communicates (via monosynchronous protocol) with the MIF and handles all communication and handshaking with the data terminal equipment (DTE) connected to the SLQ radio modem.

b. Radio Data Transfer

The SLQ, which is available in the 138-174, 406-430, 450-512, and 928-960 MHz radio frequency ranges, is designed to operate over a radio path established with another SLQ. In order to communicate properly, the transmit and receive frequencies of the units must be appropriately coordinated. The evaluation package tested in this research operated at 153.440 Mhz.

The SLQ is designed to transfer data transparently; i.e., the data applied to the transmitting SLQ is output by the receiving SLQ without alteration. No data error correction schemes - which would add overhead data and thus slow the data transfer speed - are provided by the SLQ; all information processing (i.e., addressing, CRC, forward error correction, etc.) is performed by the data terminal equipment (DTE), not the SLQ.

c. Data Transfer Reliability

The SLQ is advertised to transfer data as reliably as typical wireline modems (the rated bit error rate is 10^{-5} at an RF signal strength of -90 dBm). The receiving SLQ automatically drops the squelch tail from any incoming data signal, preventing data errors that are commonly associated with squelch noise. Also incorporated in the design is a data scrambler, which provides the necessary data transitions if a continuous string of ones must be transmitted. To prevent system lockup, a watchdog

timer monitors the operation of the SLQ every 1.4 sec and automatically resets it if a failure occurs.

d. Half- or Full-Duplex Operation

The SLQ-96 can be ordered from the factory in either a half- or a full-duplex configuration. Half duplex means that the SLQ can transmit and receive data, but it *cannot* transmit and receive at the same time. Full duplex, on the other hand, means that the SLQ can transmit and receive data at the same time. Full-duplex operation typically requires two assigned radio operating frequencies (separate transmit and receive frequencies for each unit) and a duplexer, a passive radio device that allows simultaneous transmission and reception of different frequency radio signals through the same antenna. Since data is only transmitted from the IMU end and received at the controller end, the evaluation package was configured for Half-Duplex operation.

e. Interfacing and Compatibility

Interface to the SLQ from the customer-supplied DTE is serial RS-232 via a DB-25 connector. To the DTE connected to the unit, the SLQ looks like a typical wireline modern. Character length can be set to 6, 7, or 8 bits with even, odd, or no parity. The SLQ can be configured (using a DIP switch) to communicate with the DTE synchronously or asynchronously, with internal or external transmit clocking in the synchronous mode.

The SLQ's transmitter can be set to operate in a switched mode (where the DTE controls transmitter keying) or in a continuous mode (where the transmitter is always keyed and is not controlled by the DTE). The SLQ allows selection of either a full hardware handshaking mode (where data transfer between the DTE and SLR/SLQ is controlled by handshake signals between the two pieces of equipment) or a three-wire mode (where the SLQ ignores handshaking signals and begins processing data from the DTE whenever it detects the first data character from the DTE).

The SLQ contains a 2K buffer (1K transmit data, 1K receive data) that is used during data transfer to buffer the data and provide the necessary transmit and receive

time delays (a delay of approximately 9 milliseconds is required to stabilize the transmitter before data can actually be transmitted).

f. Mechanical & Electrical Specifications

The SLQ-96 features a rugged iridited aluminum RFI enclosure that can be used in an airborne environment. Because of the SLQ's relatively compact design, the duplexer, if required, must be mounted externally. All status indicator lights are mounted internally and can be seen only when the cover is removed.

The SLQ-96 operates from a +12.5 VDC source (negative ground) and has built-in reverse polarity protection. No overvoltage protection is provided. The current drain when the modern is transmitting is 1.5 A maximum; in the receive/standby mode, it is 0.5 A maximum.

2. Datalink Setup

The setup for the RF down link using the SLQ-96 units is shown in Figure 7.2. Power on the ARCHYTAS is provided by the engine alternator. Input/output signal connections for the SLQ-96 are made via a standard DB-25 (25-pin D) connector, which is discussed in Appendix D. The transmitting SLQ-96 is onboard the UAV and is connected to the IMU via a connector cable. This cable also connects the alternator with the IMU and is shown in detail in Figure 7.3. Power to the SLQ unit is provided through a separate cable.

The receiving SLQ-96 is connected to the C30 IP_SERIAL module and the PC terminal port, through the connector cable shown in Figure 7.4. Power to the SLQ is provided via a separate power cable.

3. SLQ-96 Configuration

Configuration of the SLQ-96 units is very critical for the proper operation of the datalink and should be performed once before initialization of the system. Three banks of DIP switches (a total of 24 switches) located on the figure board of the SLQ are used to configure the operating parameters. The DIP switches, their functions and the selected

settings are covered in detail in Appendix D. For the IMU downlink, the SLQ units were configured for 9600 bps with 8 bit character length and one stop bit. Their operation was set for Half Duplex, asynchronous, three-wire mode, with 5 millisecond time out and 0 millisecond CTS delay. The transmitting unit was set for continuous Tx and switched Rx, while the receiving unit was configured for switched Tx and continuous Rx. Whenever a DIP switch is changed, the SLQ must be restarted for the change to take effect.

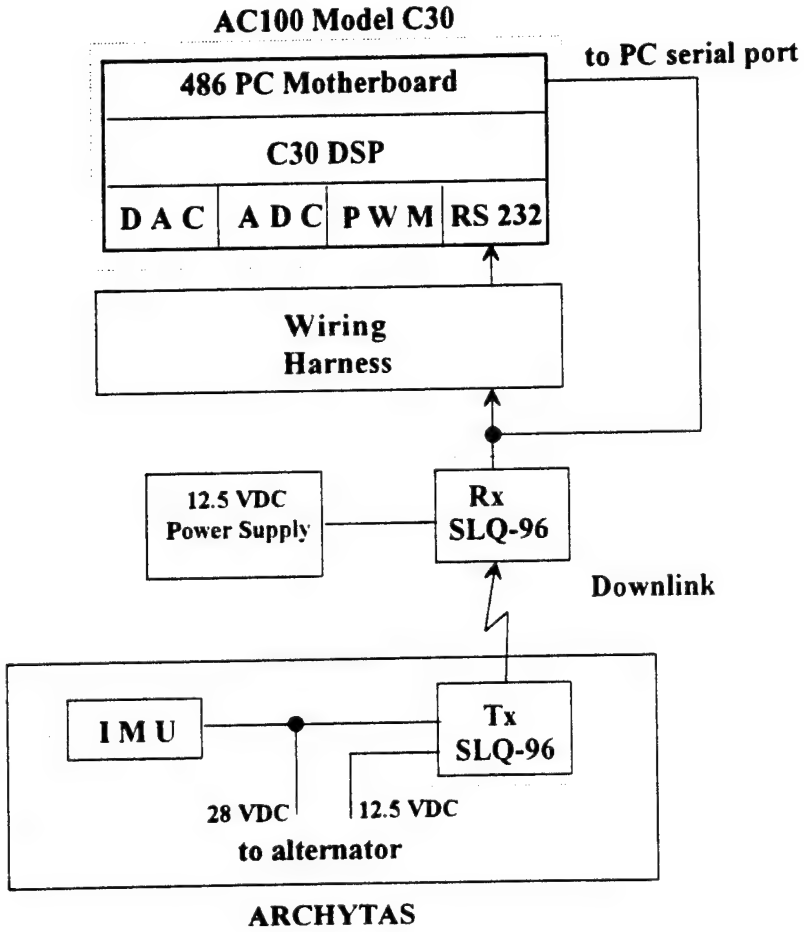


Figure 7.2: Downlink Setup

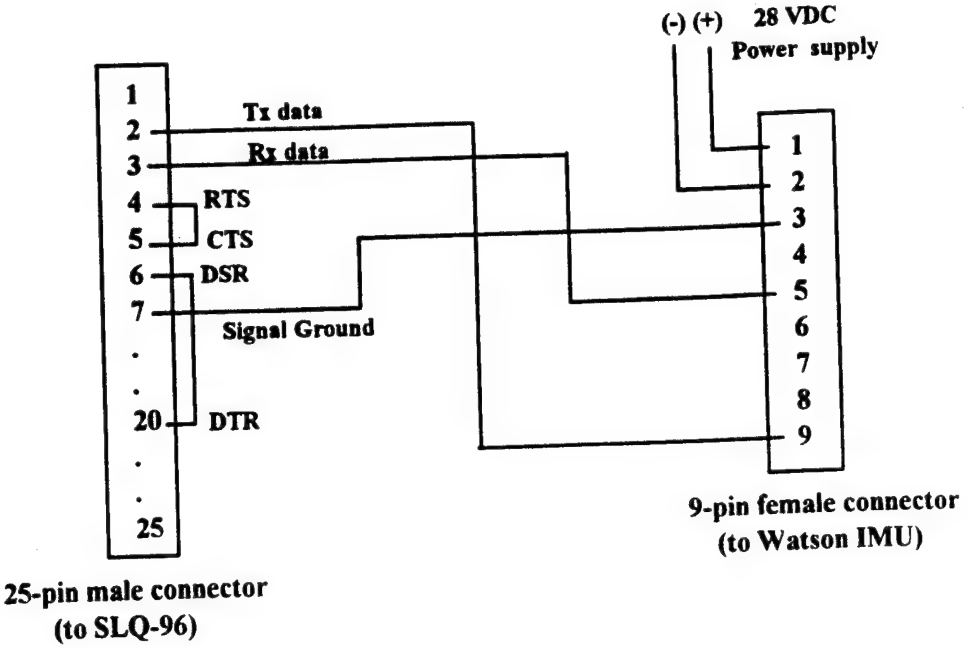


Figure 7.3: Transmitting End Connector Cable

4. Operation

Operation of the SLQ-96 system is simple and straight forward. Once the units are powered up, the link is immediately established. For proper operation it is recommended that the receiving SLQ-96 is powered before the transmitting unit. After correct functioning is established, the transmitting SLQ can be turned on and off without problems, provided the receiving unit is always on. Once the receiving SLQ is turned off, in order to reestablish the link connection the transmitting unit should also be turned off, and then turned on after the receiving SLQ is powered up. Details on the available modes of operation are discussed in Appendix D.

E. EVALUATION

Evaluation of the datalink development was achieved by Hardware-in-the-Loop (HITL) testing, following the detailed instructions of Chapter VI. Deviations from the original procedure are discussed below. The results that were obtained, indicated the validity of the implementation and are provided at the end of this section.

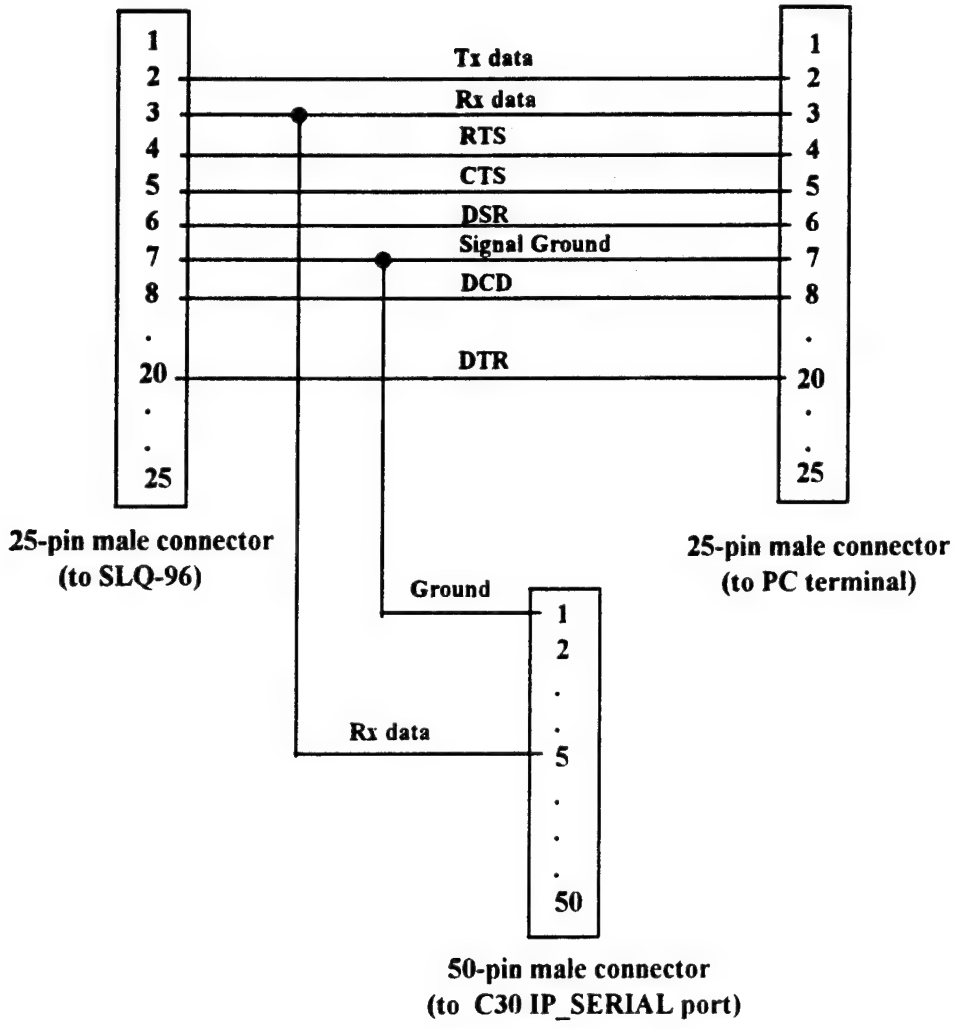


Figure 7.4: Receiving End Connector Cable

1. Setup

The complete setup for a HITL test using the AC100 Model C30 in untethered configuration is as depicted in Figure 7.5. Both the developed SLQ-96 downlink and a possible uplink are included for completeness. In untethered flight the analog actuator sensors are not connected to the AC100C30. Hardwire connection of the actuator sensors with the C30 is not required, provided calibration is performed before flight. The digital IMU is connected to the IP_SERIAL module via the receiving SLQ-96 of the RF

downlink. Commands to the servos are sent through the IP_DAC module, via the transmitting end of the RF uplink. The joystick control box is connected to the IP_ADC.

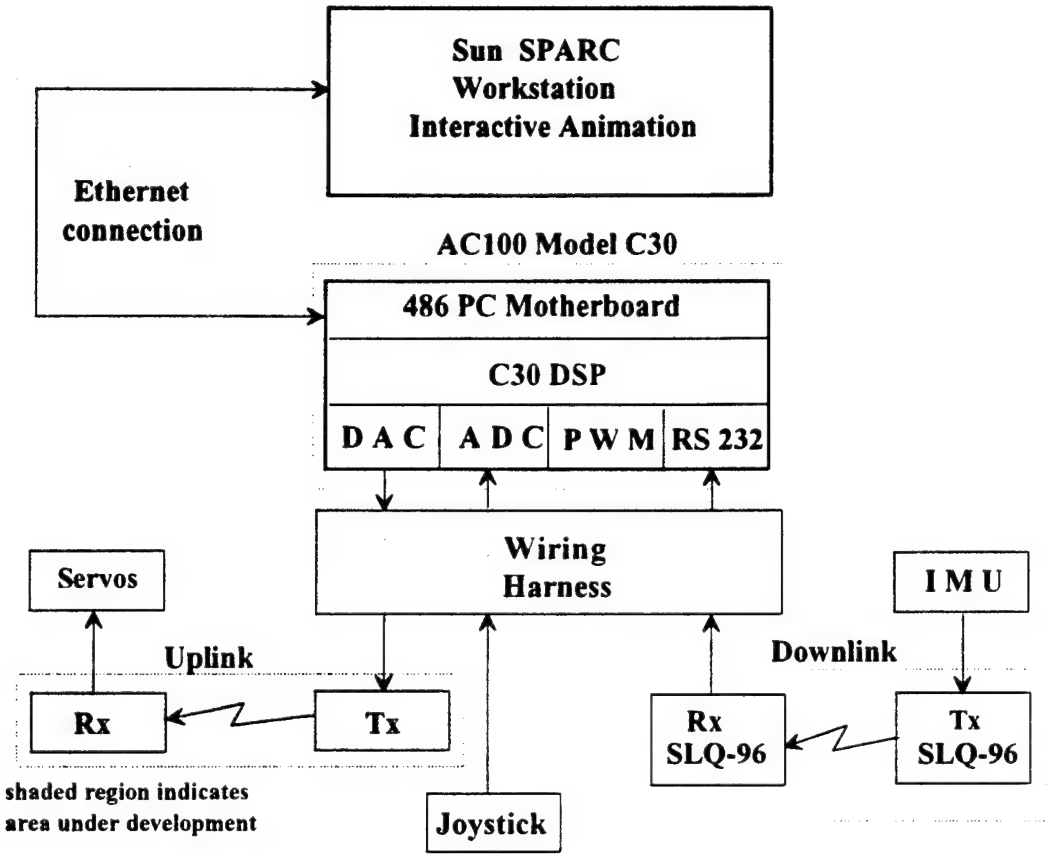


Figure 7.5: ARCHYTAS Hardware-In-The-Loop Setup, Untethered Flight

Calibration must be performed prior to HITL testing. In the untethered configuration, calibration involves measuring the sensed actuator voltage before flight, for vane positions of +/- 30 deg, and zero degrees. The difference to the values provided in Chapter VI, is dictated by the fact that in this case commands to the servos are sent from the joystick through the IP_DAC module. Full throw of the joystick controls forces the servos to deflect only to +/- 30 deg, which is more than adequate for flight, but much less than the deflection commanded by the IP_PWM module, used in the tethered case. In an attempt to make the procedure for both configurations straight forward, the calibration

screen (Figure 5.3) has been grouped and labeled according to the function performed by each switch and dial.

2. HITL Results

The HITL results were identical to those obtained and presented in Chapter VI for the tethered case, thus proving the validity of the link development and implementation. The addition of the SLQ-96 datalink system did not interfere with the controller operation. Proper reception of IMU data was verified by correct indications on the Operation IA screen (Figure 5.2) of the SystemBuild SAS project. Reception of IMU dataflow was also confirmed by a terminal program on the 486PC. Error free link operation was achieved in the lab with a separation of 20 feet between the transmitting and receiving SLQ-96 units, using non-optimal rubber antennas. Estimations on the maximum achievable range can be attempted at a later stage, during field testing and with the availability of appropriate antennas.

F. CHAPTER SUMMARY

The steps taken for the development of a wireless downlink for the ARCHYTAS have been discussed in this chapter. First the requirements for a wireless communication link to send IMU data from the UAV to the ground based controller were established. These led to the development of a dedicated RF datalink for use during untethered flight. A COTS RF system is selected for implementation, since previously tested units cannot meet the present criteria. The validity of the datalink development is verified by meeting design requirements, and by obtaining the same HITL test results with similar tests performed in tethered configuration.

VIII. COMMENTS AND CONCLUSIONS

This primary goal of this thesis was the successful integration of a functional Inertial Navigation System (INS) with the ARCHYTAS Stability Augmentation System (SAS) hardware and software. Employing the available tools and components, requirements were set and satisfied, through analysis and modification of existing hardware and software. The steps taken for incorporation of the INS into the SAS included modification of the already existing flight controller, development of a new SystemBuild flight controller project, interfacing of the IMU with the AC100C30 with appropriate hardware and software modifications, and installation of the sensor on the platform. Furthermore, this research examined the development of a wireless radio frequency (RF) data link, that transmits IMU data from the UAV to the ground control station and will be employed during the untethered stages of flight. Testing and evaluation of the integrated SAS, during tethered and wireless modes of operation provided proof for the validity of the research.

A. ACCOMPLISHMENTS

From the general goal to "design an INS suite and successfully implement it with a real time controller for autonomous flight of the ARCHYTAS UAV", specific requirements were derived. From these operational requirements, hardware components were selected, software was modified and procedures were specified. In the course of the design and implementation of the INS with the SAS, several significant milestones were achieved:

- The data (and its format) required by the SAS to perform flight control of the ARCHYTAS were determined.
- An Inertial Measurement Unit (IMU) was selected and configured according to the controller requirements.

- The ARCHYTAS SAS software was modified for integration with the IMU and the control joystick. A new SystemBuild project was developed and tested.
- Hardware integration of the flight controller with the IMU and joystick was successfully implemented and tested for correct HITL operation.
- The requirements for a wireless communication link to tranfer data from the airborne IMU to the ground control station, were determined and satisfied.
- An off-the-shelf RF data link system was selected according to the requirements.
- The down link for the IMU was successfully developed and tested for correct HITL operation.

The ARCHYTAS SAS was modified to be integrated with the IMU, based on the controller already designed and implemented in previous researches [Ref. 3, 5]. Much attention was focused on development of serial drivers for trouble free data flow from the IMU to the AC100 Model C30. Employment of all available software tools provided for user friendly operation in all available modes. The wireless down link for transmission of data from the airborne IMU to the AC100C30 was developed for future untethered flight stages. A representative RF system satisfying the determined requirements was selected and tested with successful results.

This thesis research details the design criteria and the procedure for the incorporation of an INS with the already developed ARCHYTAS flight controller, to give the reader a better understanding of the overall project. From this understanding, present design decisions become apparent, while future developments and modifications are facilitated. Recommendations for future improvement are provided below, in an attempt to assist the development of a more efficient SAS for the ARCHYTAS UAV.

B. RECOMMENDATIONS

In the course of this thesis research, it became evident that several steps and improvements would be necessary for further refinement of the complete ARCHYTAS SAS. These include system tests that were not completed because of time limitations, improvements to SAS software and hardware that became evident during development and testing, as well as general system modifications. These areas are recommended for future research and are briefly delineated below.

1. System Tests

Because of numerous hardware problems and troubleshooting during the initial engine tests on the thrust stand, the time of original flight testing was significantly delayed. By the time this research was completed, only HITL testing of both tethered and wireless configurations were completed, with successful results. Altough it was intended by the author to perform flight tests during this thesis work, the tests outlined below should be performed in future research.

For the ARCHYTAS, the flight test phase is intended to be performed in three stages:

- Test Stand
- Tethered Flight
- Actual Flight

The equipment required for (tethered) flight testing is shown in Figure 8.1, while the procedure for moving these components outside the lab, is described in Appendix C.

a. Test Stand

The first test would be conducted in a laboratory facility on a test stand, (Figure 8.2) which would allow some degree of movement, while restricting free flight so that the vehicle can not be damaged. The scope of this test would be to evaluate with reasonable safety, the flight control algorithm and implementation during restricted

hovering. Before flight, calibration would be performed. Once the ARCHYTAS engine is started, the SAS would employ the aircraft model for the ARCHYTAS. Then the operator would increase the throttle, to lift the vehicle off the ground. Normally, the SAS would then switch to use of IMU data for stabilization of flight. At that point, provided the controller is well designed, the vehicle would have to behave properly to operator commands, maintaining stable flight conditions, and absence of roll. Restricted response to commanded angles and rates can be tested and verified, within the 2-3 ft of allowable movement in each direction. Further tests would have to be performed as determined by the controller behavior. This would be the time when full confidence on operator ability and SAS functioning would have to be acquired, before proceeding to the next stage.

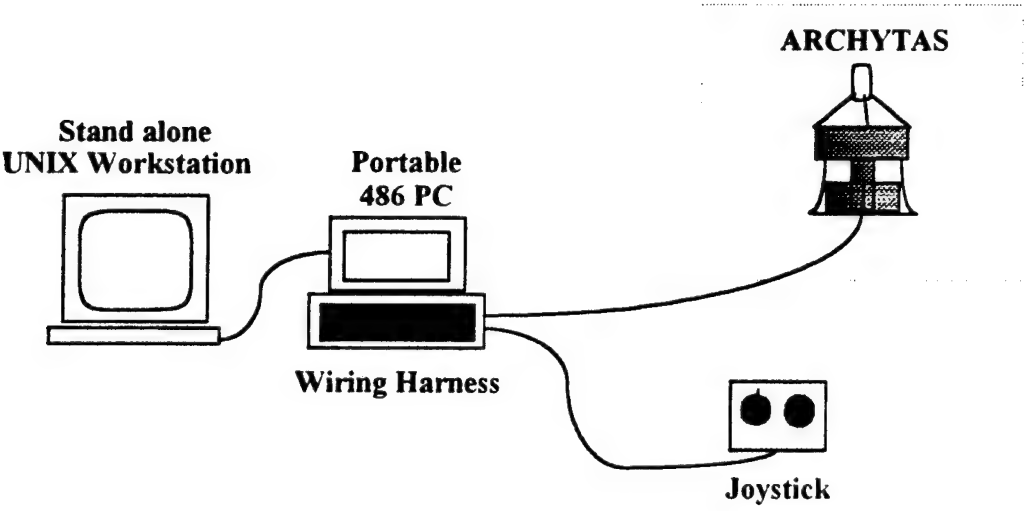


Figure 8.1: Flight Test Setup

b. Tethered Flight

The next stage is tethered flight. The ARCHYTAS would be evaluated in three degree of freedom flight, attached only to the ground control station by means of the tether. An experienced pilot would be commanding the aircraft with a joystick, while manual override would be available for direct manual control of the vehicle, in the event of a problem. Tethered free-flight tests would be conducted to verify operation of control

laws and the SAS. For this test, the component setup would be as depicted in Figure 8.1. During this phase, extended maneuvers as well as change of position and altitude can be performed.

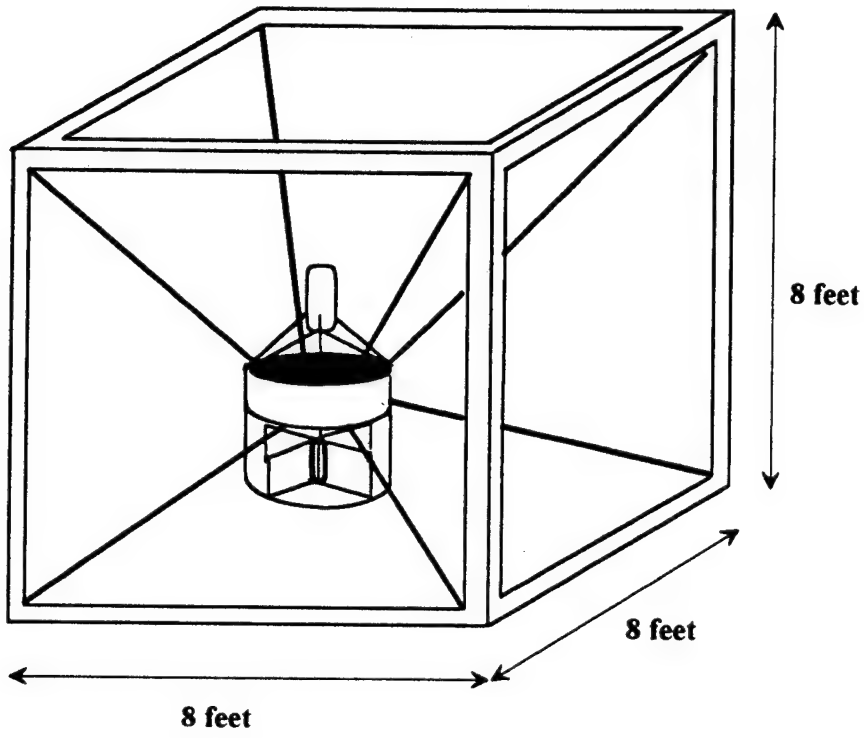


Figure 8.2: Test Stand

c. Actual Flight

The final and ultimate test of the control design is the untethered autonomous flight test. Once the previous stages are successfully completed, the ARCHYTAS can be evaluated in three degree of freedom free flight, communicating with the ground control station by means of radio frequency (RF) up and down data links. Commands can be given either by the joystick or by monitor inputted values. Equipment

setup can be as shown in Figure 8.3.

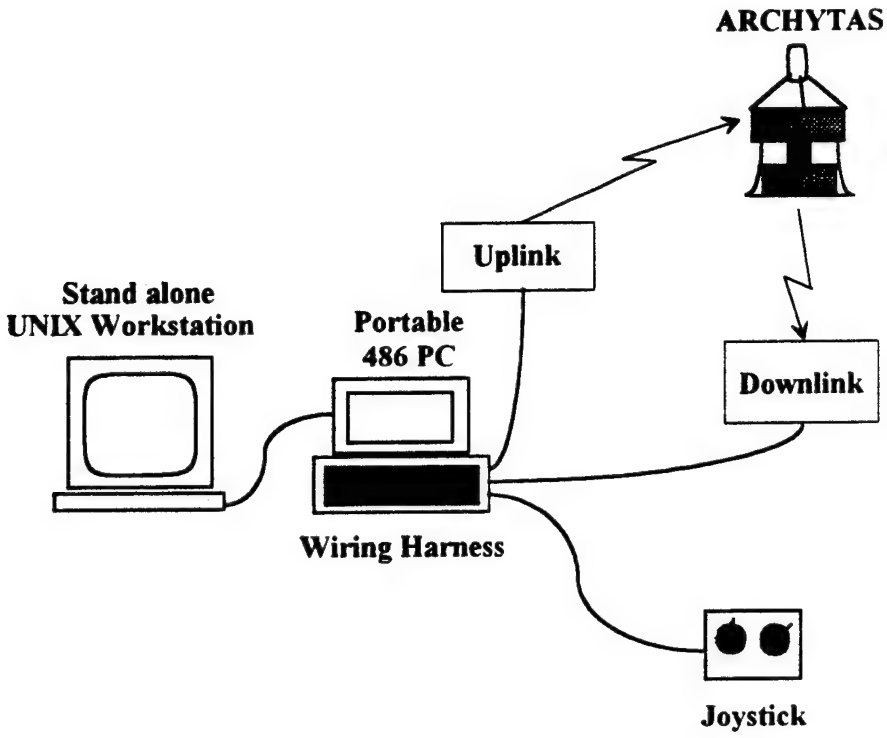


Figure 8.3: Actual Flight Setup

2. SAS Software Recommendations

The serial buffer overflow messages that occur before the start button is pushed during the SystemBuild flight controller operation, can be eliminated. The reason for the problem is that the interrupts are turned on early in the program. A possible solution is to edit the file 'ipserial.c'. To achieve this, look for the line that begins, 'unmask\dot'. This line should be moved to the user defined initialization section, so that the interrupts will be turned on at an appropriate time.

3. SAS Hardware Recommendations

Having encountered numerous problems during early engine runs, it was determined that all equipment should be mounted using vibration damping techniques and

fixtures. Before actual flight test of the vehicle, all hardware should be secured to prevent screws and nuts from backing off due to vibration.

As far as wiring problems are concerned, there are several possible solutions that need to be addressed, prior to any in flight testing of the vehicle. One of them is to use a smaller gauge (thicker) wire when soldering connections. The wires will then be able to better withstand the continuous engine vibrations. Another solution is to coat the connections in a type of silicon, similar to that used in the ignition circuits. This will reinforce any connections, but will make maintenance slightly more difficult. A third solution is to provide foam padding internally, to absorb the vibrations while restricting movement of the plugs.

Finally, cannon plugs can be hard mounted within the body of the ARCHYTAS itself. This will allow minimal movement of any parts and provide the greatest strength. However, it will be the least flexible configuration as far as maintenance is concerned.

4. General System Recommendations

The effect of noise from the ARCHYTAS ignition system on the RF link during normal operation, needs to be tested. The use of an EM shielded casing for the airborne unit should be examined.

For future incorporation of radio transceivers and flight sensors, a full power budget will be required, in order to determine complete present and future power requirements and margins.

Prior to regular flight testing of the UAV in untethered mode, an operating license issued by the FCC must be obtained in coordination with frequencies allocated by the Navy for UAV testing. During this research, operation and testing of the radio links was performed in the lab and FCC regulations were not violated.

In future attempts to select a different radio down-link for the IMU data, the baud rate limitations should be observed, since the data rate is directly linked to the operating speed - and eventually the stability - of the controller. Radio packet technology is

unacceptable, and three wire communication datalinks seem to be the only available commercial systems that satisfy the requirements.

For security of all the effort already invested in the UAV project, a backup system should be instituted for the stand alone workstation of the flight test setup.

C. SUMMARY

Through this research, the goal of designing and implementing an INS for real time flight of the ARCHYTAS UAV has been successfully completed. The resulting aggregation of hardware and software represents a functional shell to which improvements can be made, and into which other subsystems, developed in the future may be added. From the initial research goals, down to the final working implementation, this thesis quantifies the system mandates and documents the conceived solutions. The Stability Augmentation System integrated with the Inertial Measurement Unit represents a tested system that is ready to control the ARCHYTAS in stable autonomous flight.

APPENDIX A. MATRIX$_X$ - SystemBuild BLOCK DIAGRAMS

The following SuperBlocks are contained in the SystemBuild project for the ARCHYTAS Stability Augmentation System (SAS), currently named 'flight_test_1'. For futher details of the SAS architecture refer to Chapter V.

- **actuators:** Figure A.1 shows the SuperBlock containing 4 actuator blocks, one for each vane.
- **actuator_1:** This SuperBlock, depicted in Figure A.2, shows the actuator model for the first vane. The other vane actuators are identical to this one.
- **actuator_2:** Same as above, not shown.
- **actuator_3:** Same as above, not shown.
- **actuator_4:** Same as above, not shown.
- **ang_velocity_eq:** Figure A.3 presents the SuperBlock corresponding to the angular velocity equations. The values for the body and rotor inertia tensors and matrices are listed in Table 2.1.
- **control_process:** Figure A.4 depicts this SuperBlock corresponding to the controller part of the project. It contains the 'dcont_wind' and 'vane4x' SuperBlocks
- **dcont_wind:** Figure A.5 contains the discrete controller SuperBlock for the ARCHYTAS.
- **feedback_laws:** Figure A.6, contains the Delta implementation of the PID controller for the ARCHYTAS.
- **filters:** Figure A.7 contains four discrete Butterworth low-pass anti-aliasing filters. The necessity and the values for the vane filters, are discussed by Moats in [Ref. 7].
- **flight_test_1:** Figure A.8 shows the top level SuperBlock for the complete ARCHYTAS SAS. The inputs and outputs from this SuperBlock are listed in the 'flight_test_1.rtf' file and used in the Hardware Connection Editor.
- **imu_in:** Figure A.9 shows this SuperBlock which receives data from the serial driver and provides them to the system.
- **imu_logic:** Figure A.10 depicts this block, which contains the part of the SAS, relevant to the integration of the IMU.

- **input_to_model:** Figure A.11. The algebraic block converts the vane signals to command signals in radians.
- **input_to_vane_servos:** Figure A.12. The saturation block limits the throw of the vanes to +/- 15 degrees. The algebraic block converts the command signals to vane signals in degrees.
- **integ_ang_vel:** Figure A.13. Contains the integrators for the angular velocities.
- **integ_lin_vel:** Figure A.14. Contains the integrators for the linear velocities.
- **integ_sim:** Figure A.15. This SuperBlock contains the SuperBlocks 'integ_ang_vel', 'integ_lin_vel' and 'int_ang_sim'. Each of these SuperBlocks contains three discrete integrators, with the appropriate initial values. The 'int_ang_sim' SuperBlock also contains two sum blocks, to add in the pertubations for θ and ψ.
- **int_ang_sim:** Figure A.16. Contains the integrators for the aircraft angles.
- **joystk:** Figure A.17. Contains the 'rpm_command', 'yaw_command', 'pitch_command' and 'rollrate_command' SuperBlocks. Converts actual joystick movements to voltages.
- **joystk_logic:** Figure A.18. This SuperBlock contains the part of the SAS, relevant to the joystick integration.
- **kinematics:** Figure A.19 presents the kinematics SuperBlock, which contains the 'lin_velocity_eq', 'ang_velocity_eq', and 'L_dot_eq' SuperBlocks.
- **l_dot_eq:** Figure A.20. This SuperBlock is an implementation of Equation 4.24.
- **l_m_n_compute:** Figure A.21 depicts the calculation of the angular momentum terms. Blocks 5, 6, and 98 contain the appropriate stability derivatives, and block 7 includes the moment due to propeller thrust, given by Equation 4.5.
- **lin_velocity_eq:** Figure A.22 contains the equations for the linear forces acting on the aircraft. The 'T_value' SuperBlock contains Equation 4.4 and block 95 is the force due to gravity.
- **pitch_command:** Figure A.23. This SuperBlock contains the deadband and threshold blocks, for conversion of joystick movement to voltage.
- **plant_kinematics:** Figure A.24. Contains the 'Integ_sim' and 'kinematics' SuperBlocks, as well as a block for workspace variables.

- **rollrate_command:** Figure A.25. This SuperBlock contains the deadband and threshold blocks, for conversion of joystick movement to voltage.
- **rpm_command:** Figure A.26. This SuperBlock contains the deadband and threshold blocks, for conversion of joystick movement to voltage.
- **T_value:** Figure A.27. This SuperBlock contains Equation 4.4.
- **vane1_processing:** Figure A.28 presents the SuperBlock for vane one. It is identical to the other three SuperBlocks. Individual block functions are discussed in Chapter V.
- **vane2_processing:** Not shown.
- **vane3_processing:** Not shown.
- **vane4_processing:** Not shown.
- **vane4x:** Figure A.29 depicts the 'vane4x' SuperBlock, which contains the 'filters' SuperBlock and one SuperBlock for each of the four vanes. See Chapter V for further explanation.
- **yaw_command:** Figure A.30. This SuperBlock contains the deadband and threshold blocks, for conversion of joystick movement to voltage.

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext. Inputs	Ext. Outputs	Enable
actuators	0.0250	0.	5	4	Parent

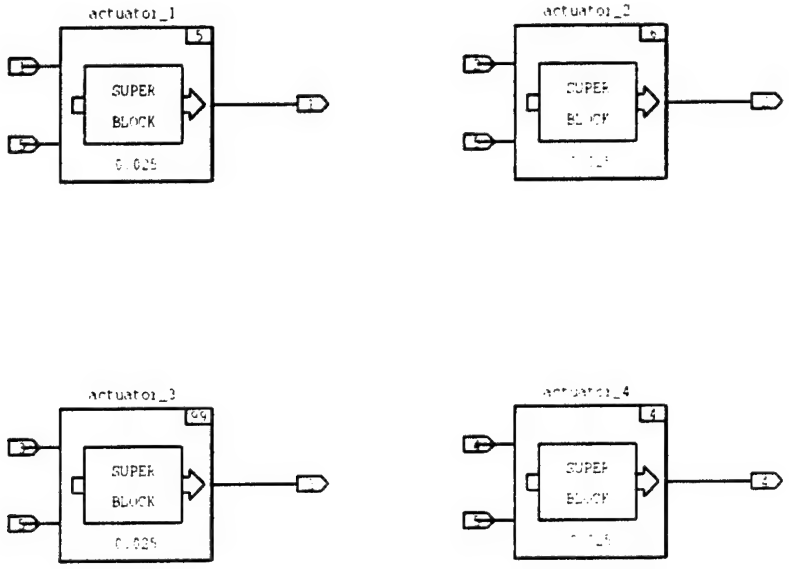


Figure A.1: 'actuators' SuperBlock

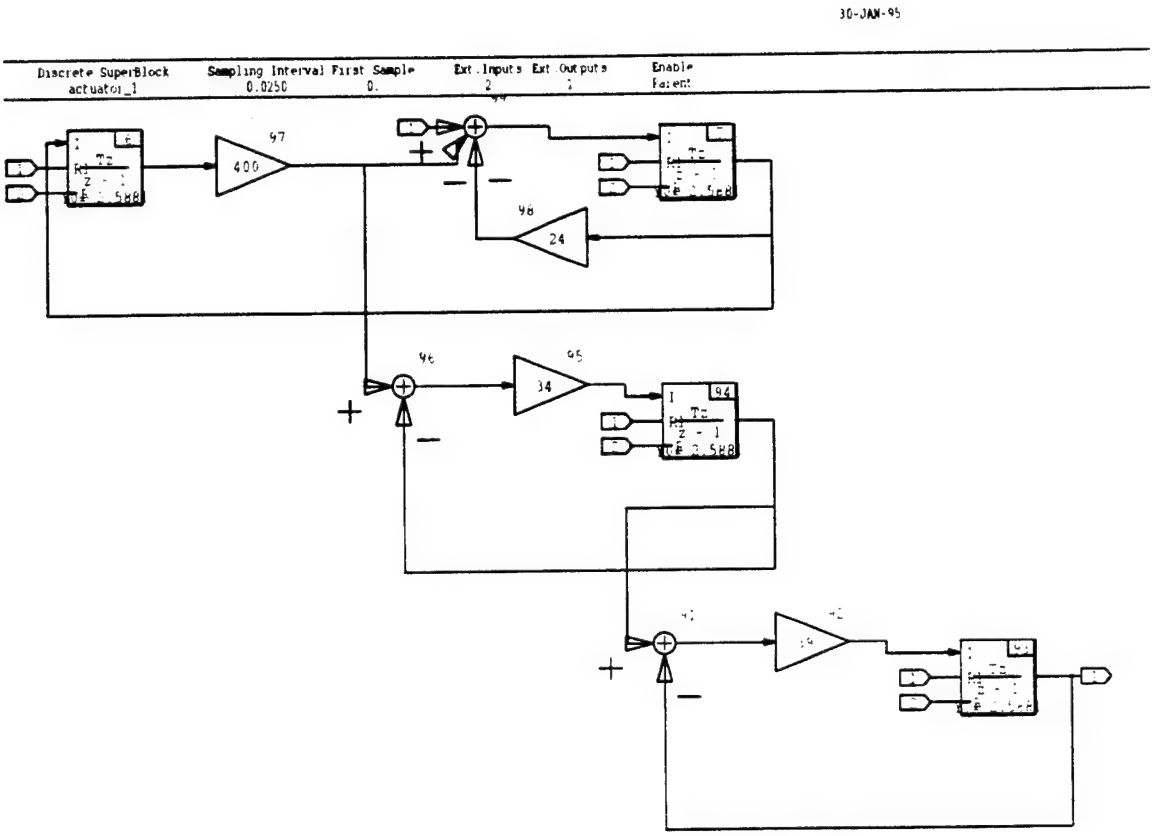


Figure A.2: 'actuator_1' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext.Inputs	Ext.Outputs	Enable
ang_velocity_eq	0.0250	0.	10	3	Parent

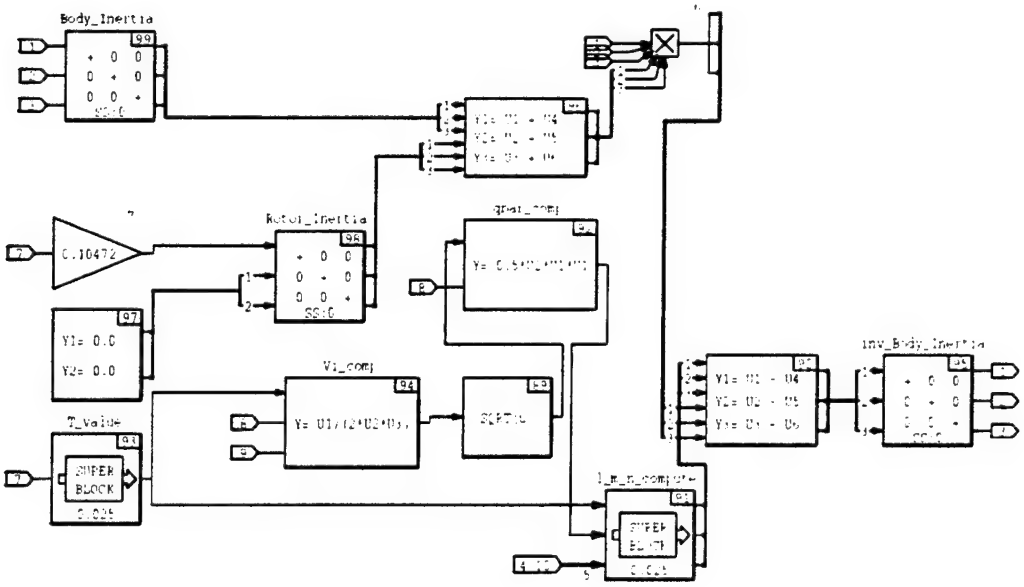


Figure A.3: 'ang_velocity_eq' SuperBlock

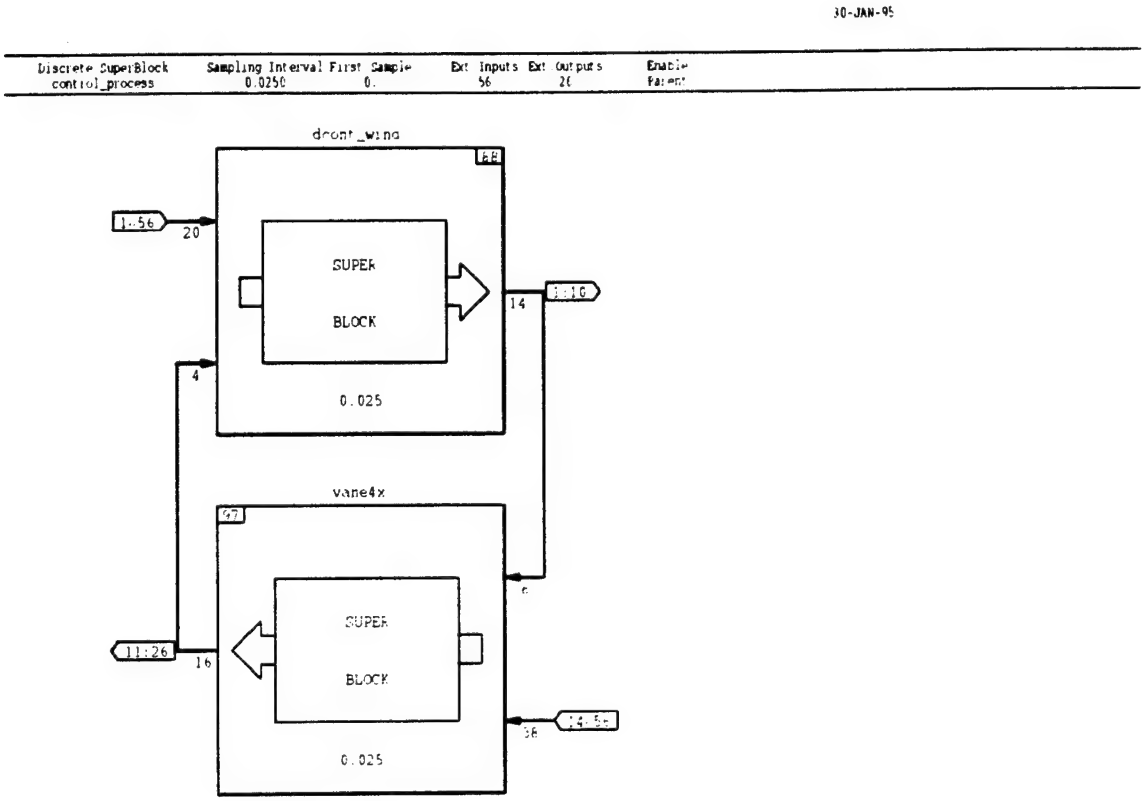


Figure A.4: 'control_process' SuperBlock

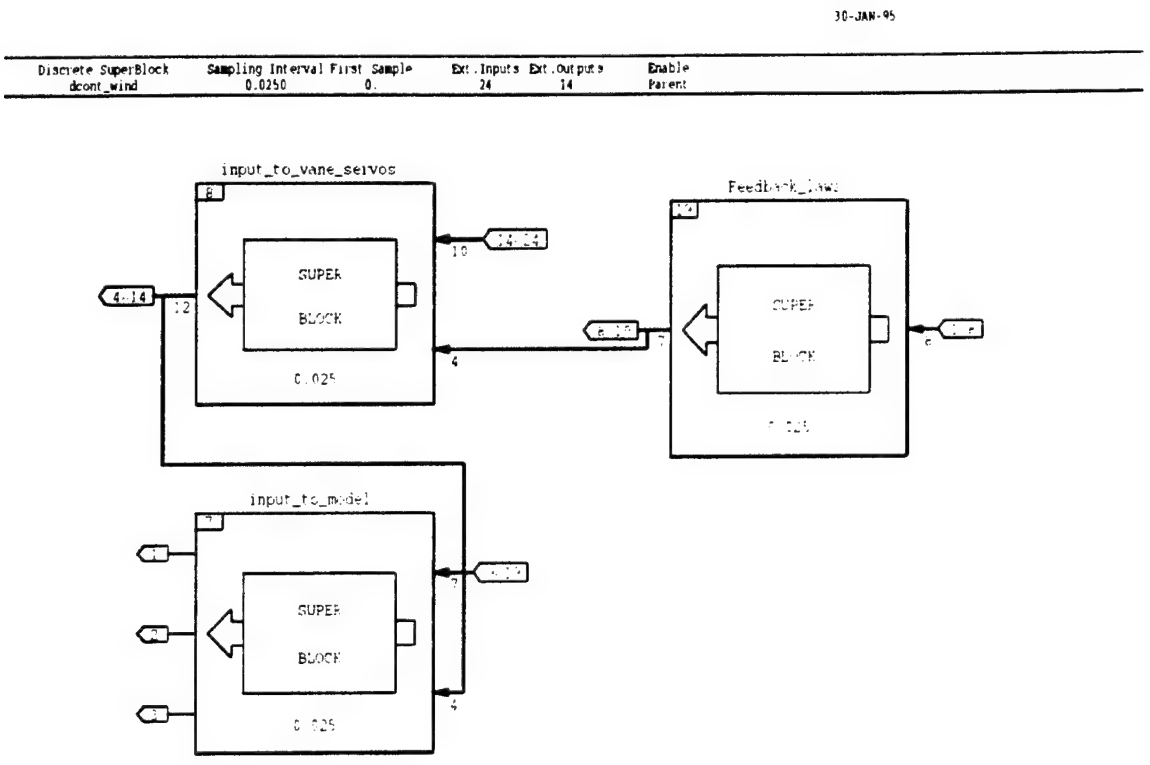


Figure A.5: 'dcont_wind' SuperBlock

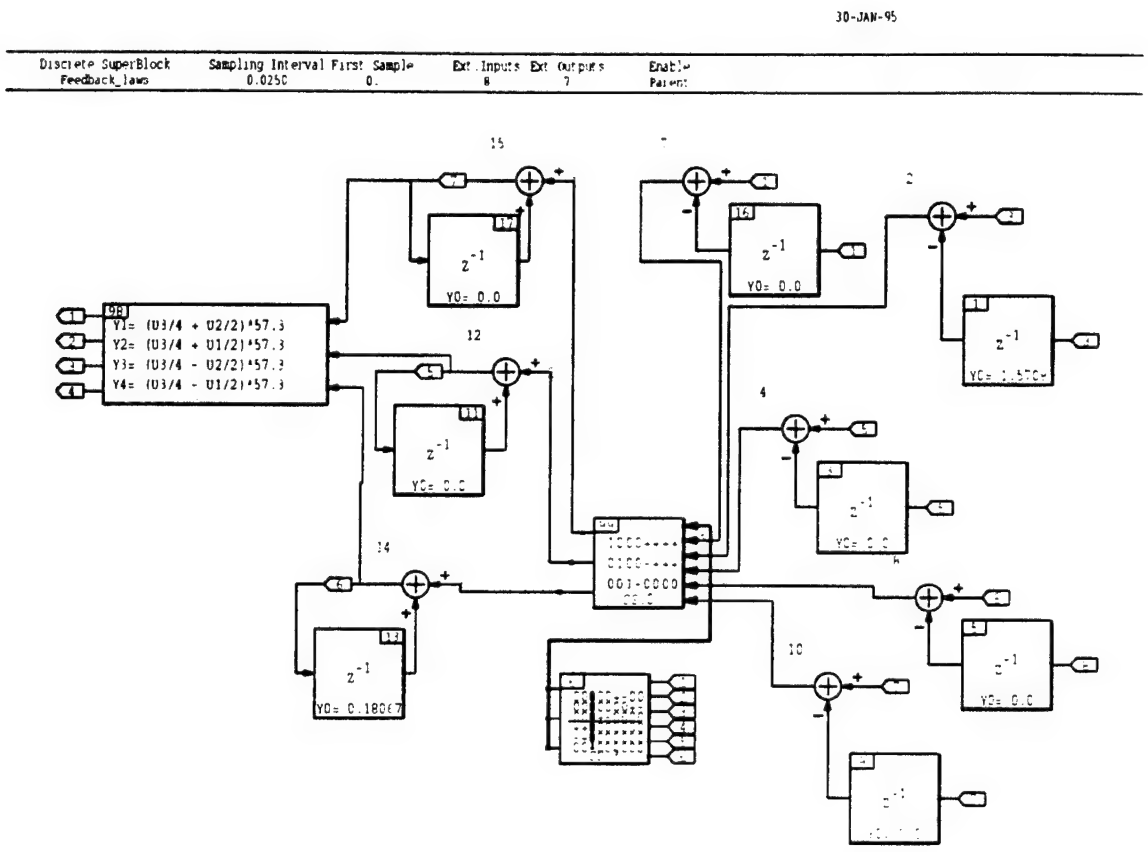


Figure A.6: 'feedback_laws' SuperBlock

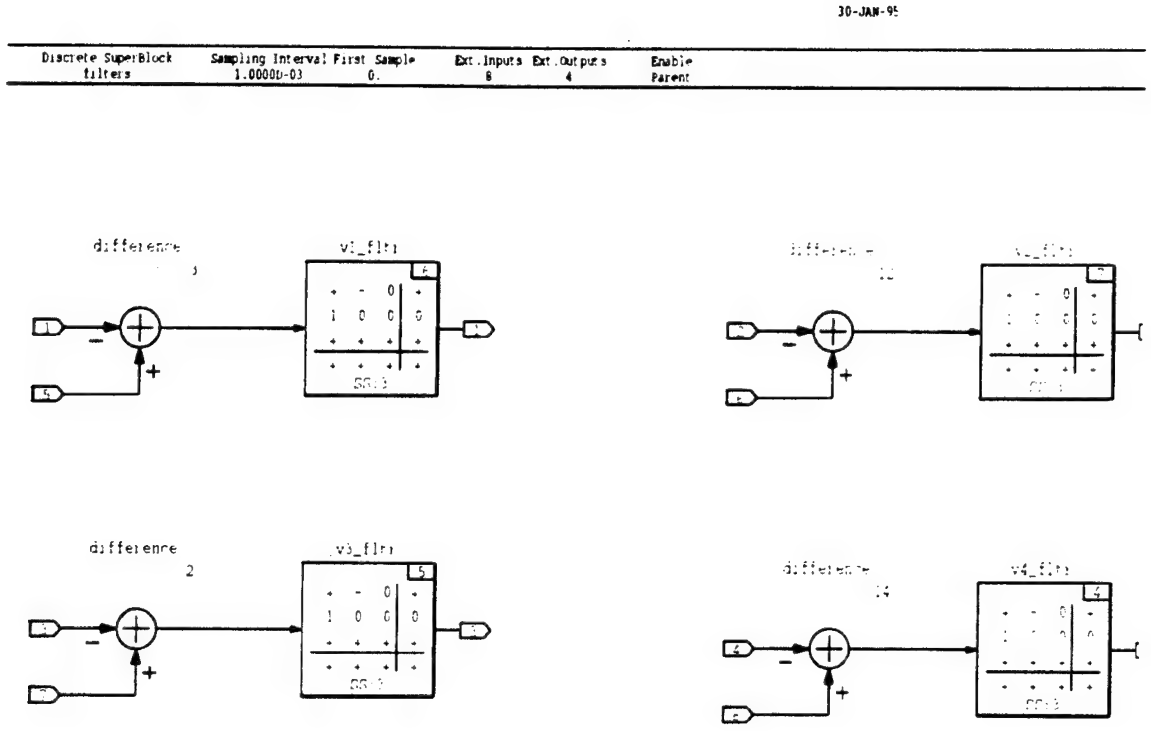


Figure A.7: 'filters' SuperBlock

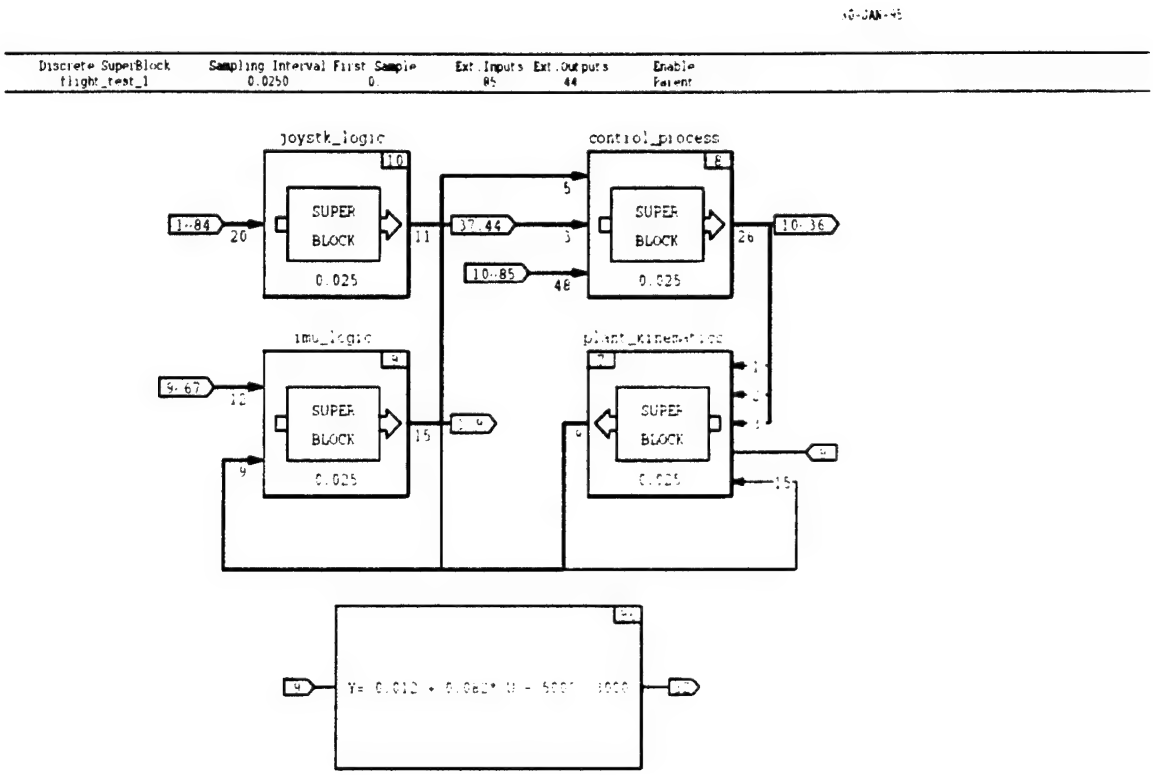


Figure A.8: 'flight_test_1' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext. Inputs	Ext. Outputs	Enable
imu_in	0.0400	0.	9	9	Parent

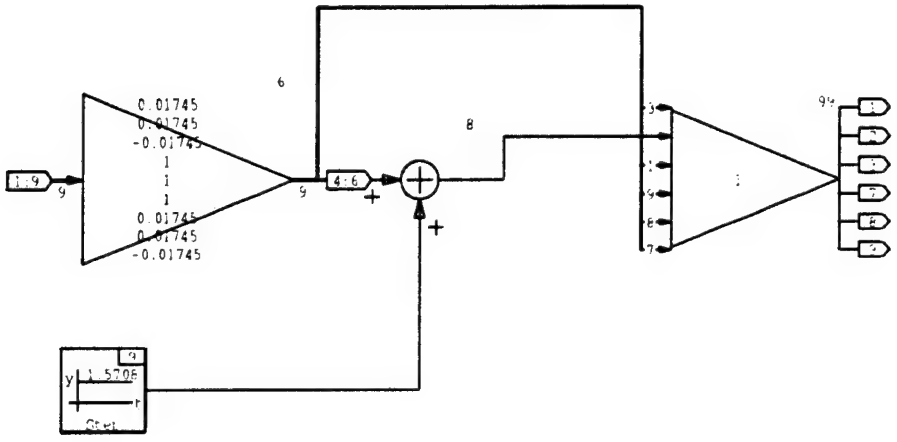


Figure A.9: 'imu_in' SuperBlock

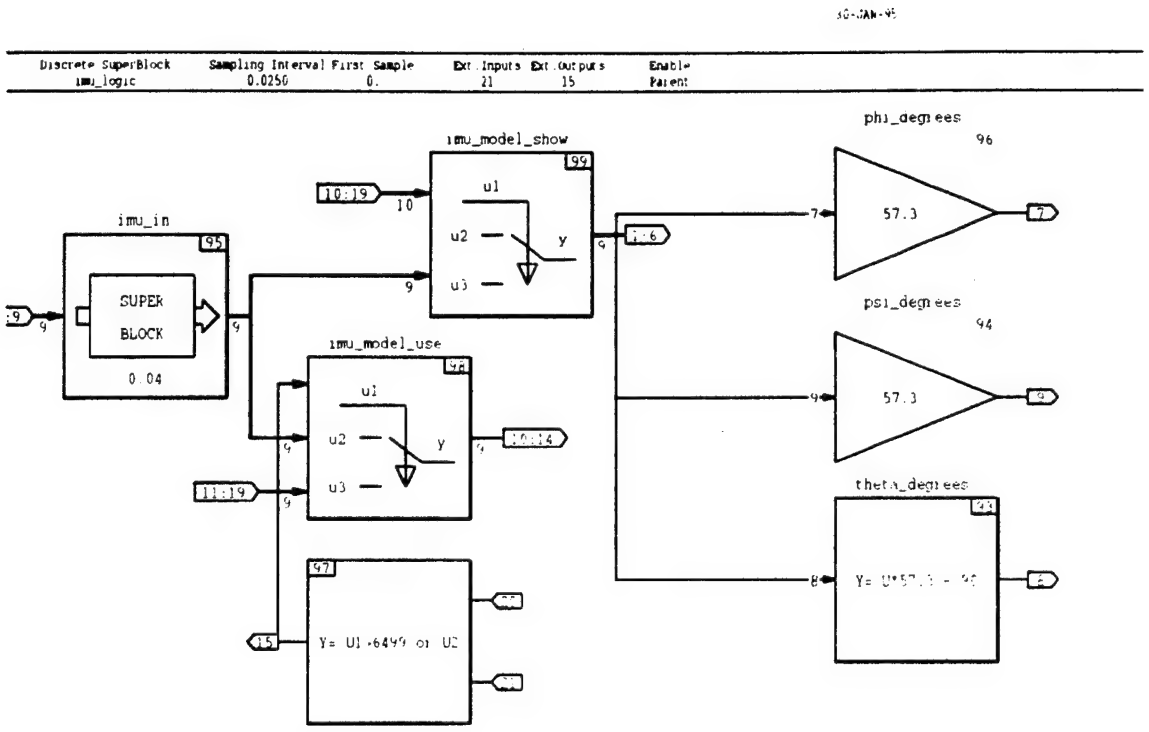


Figure A.10: 'imu_logic' SuperBlock

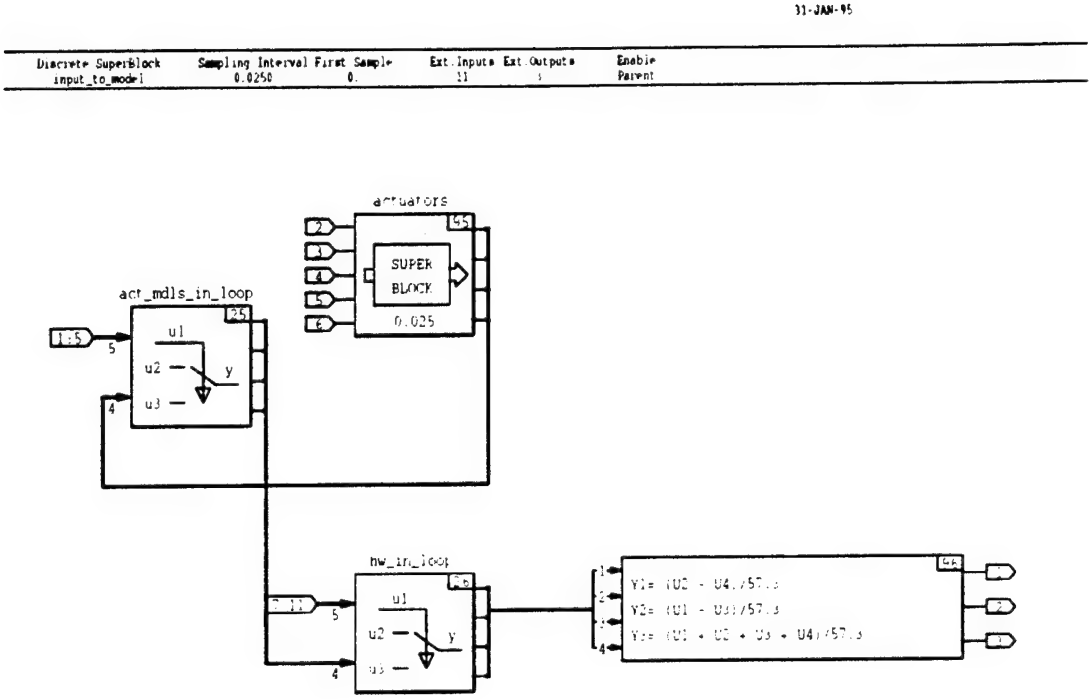


Figure A.11: 'input_to_model' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext. Inputs	Ext. Outputs	Enable
input_to_vane_servos	0.0250	0	14	12	Parent

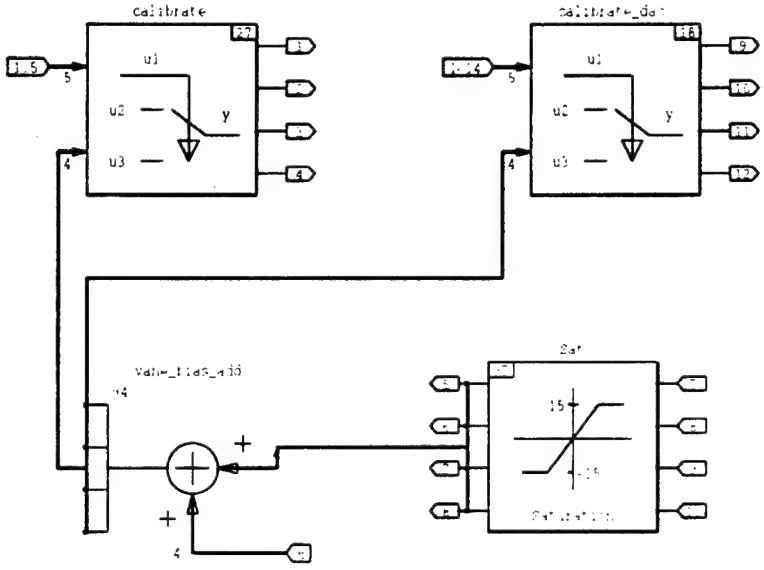


Figure A.12: 'input_to_vane_servos' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext. Inputs	Ext. Outputs	Enable
integ_ang_vel	0.0250	0.	4	3	Parent

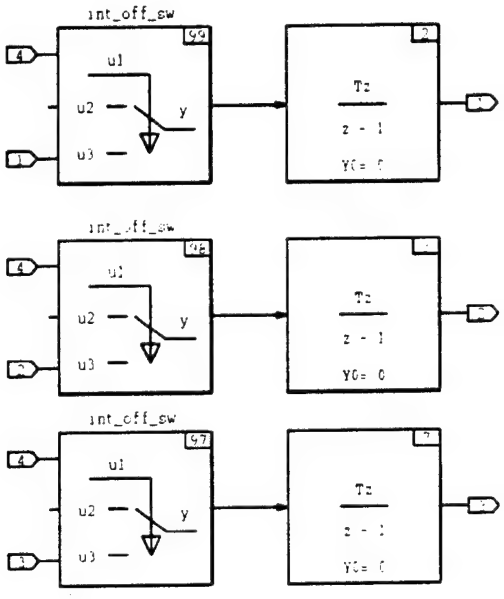


Figure A.13: 'integ_ang_vel' SuperBlock

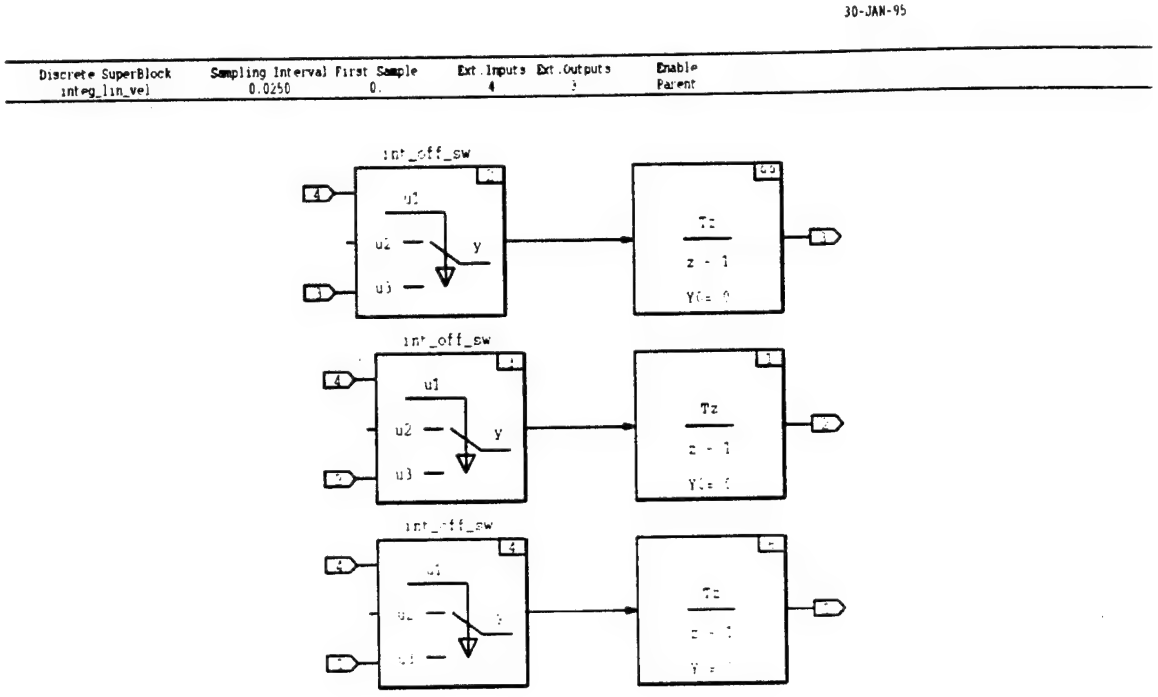


Figure A.14: 'integ_lin_vel' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext. Inputs	Ext. Outputs	Enable
Integ_sim	0.0250	0.	10	9	Parent

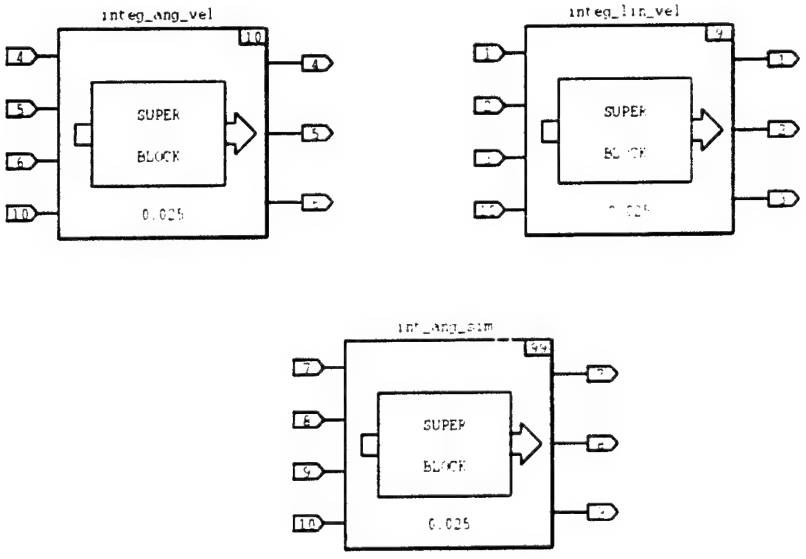


Figure A.15: 'integ_sim' SuperBlock

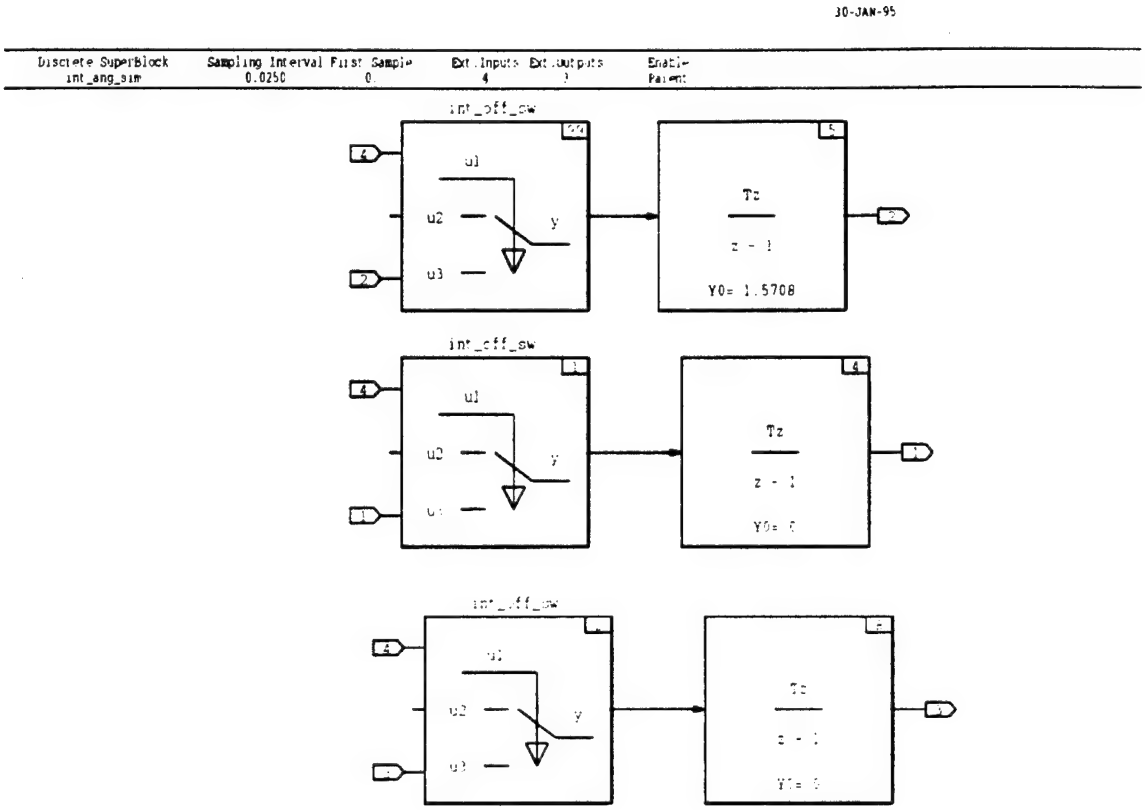


Figure A.16: 'int_ang_sim' SuperBlock

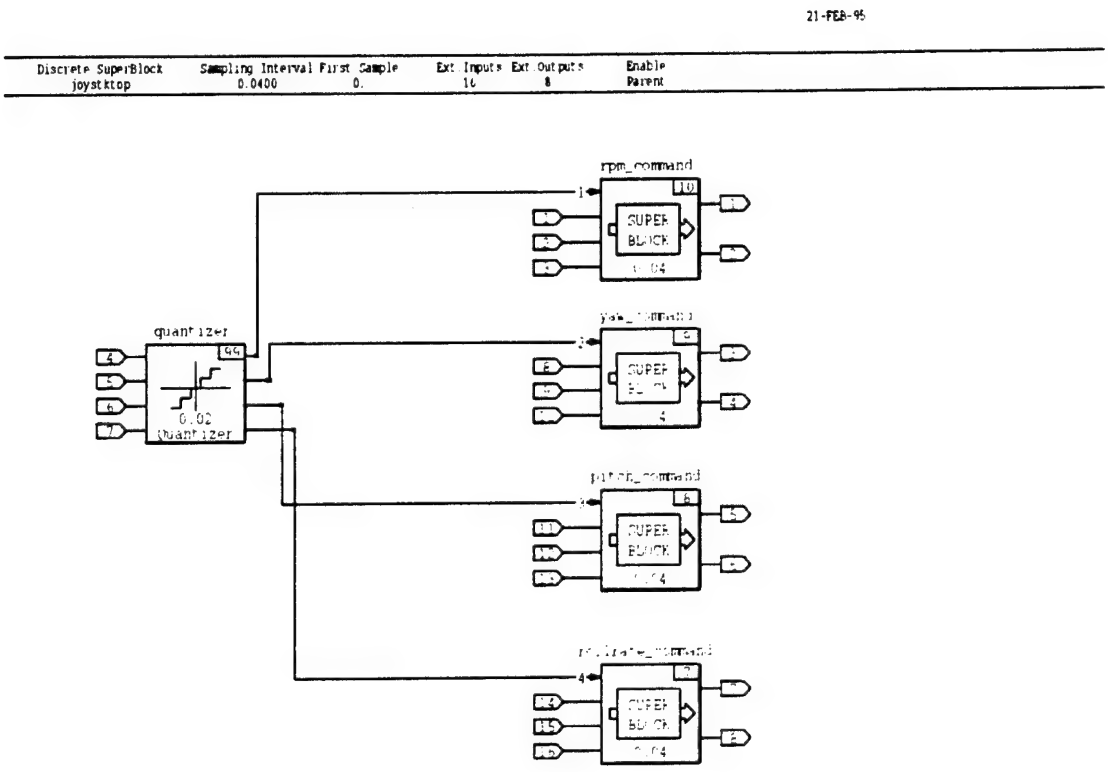


Figure A.17: 'joystk' SuperBlock

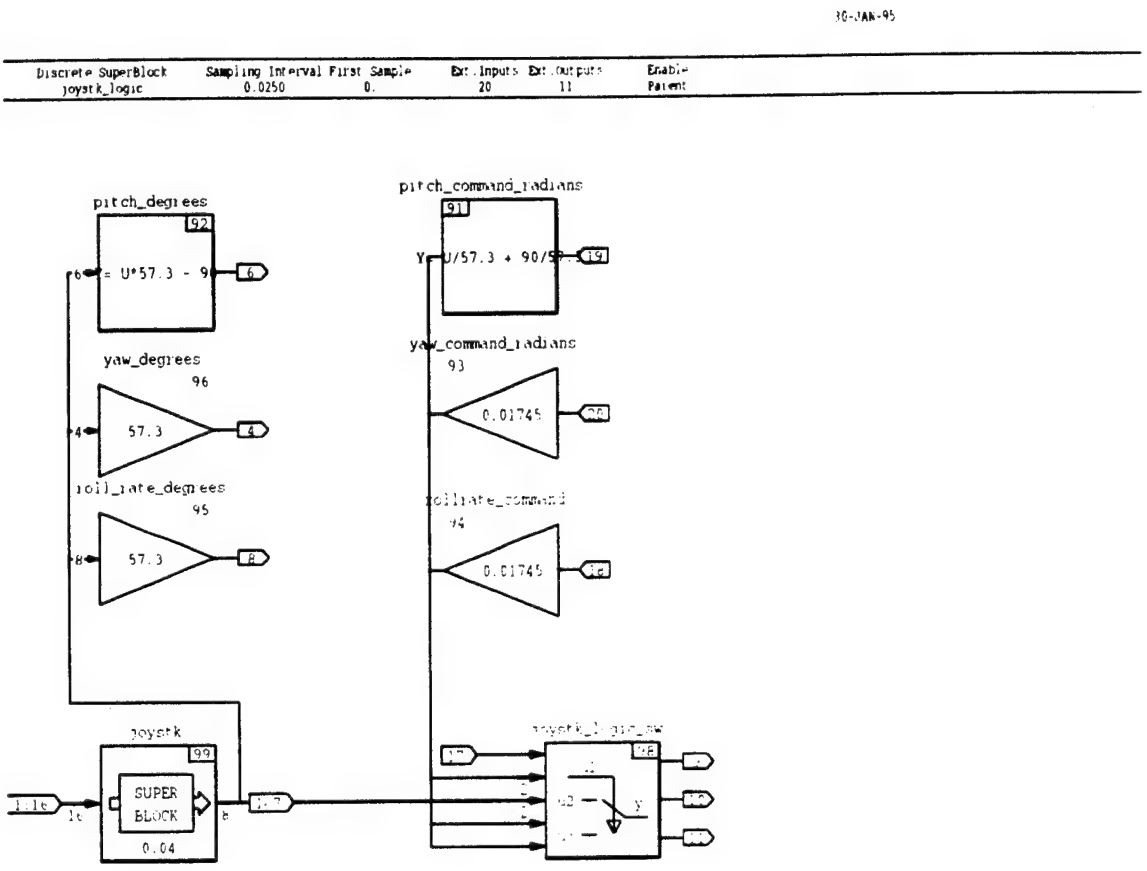


Figure A.18: 'joystk_logic' SuperBlock

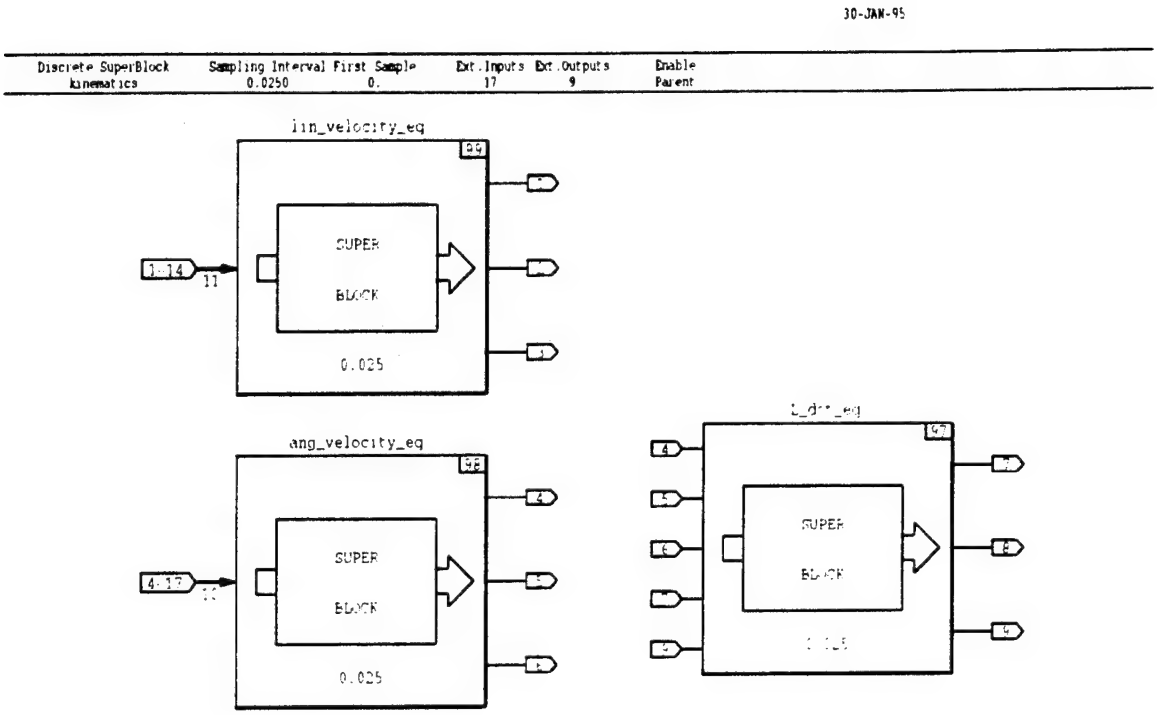


Figure A.19: 'kinematics' SuperBlock

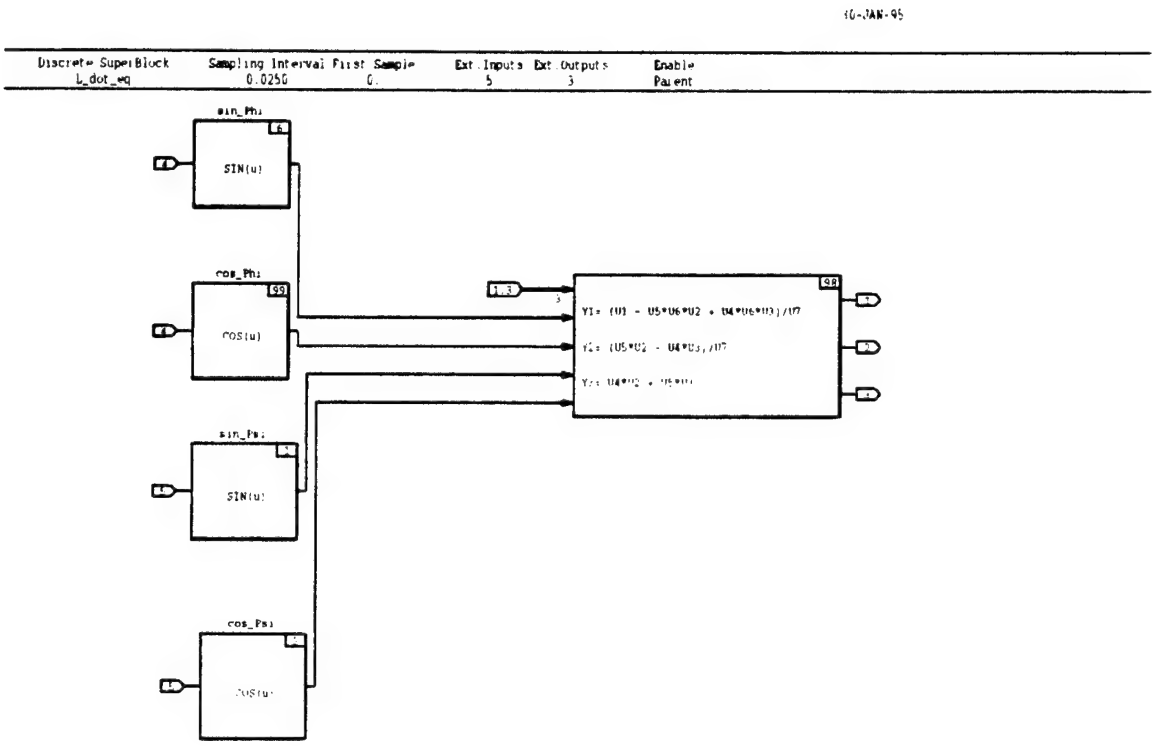


Figure A.20: 'L_dot_eq' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext.Inputs	Ext.Outputs	Enable
l_m_n_compute	0.0250	0.	7	3	Parent

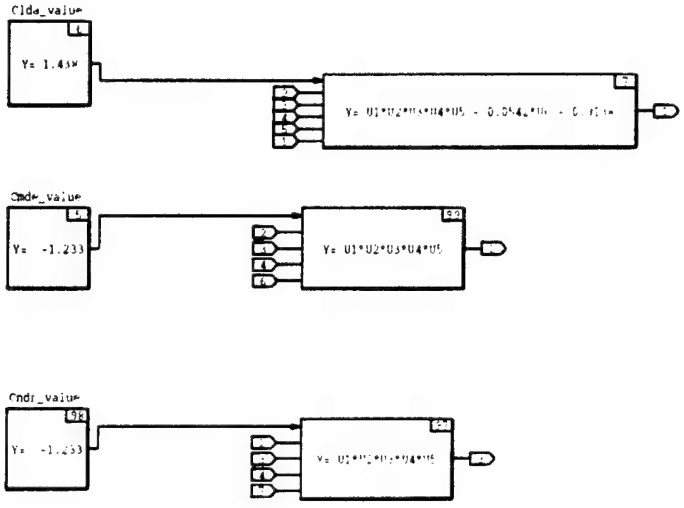


Figure A.21: 'l_m_n_compute' SuperBlock

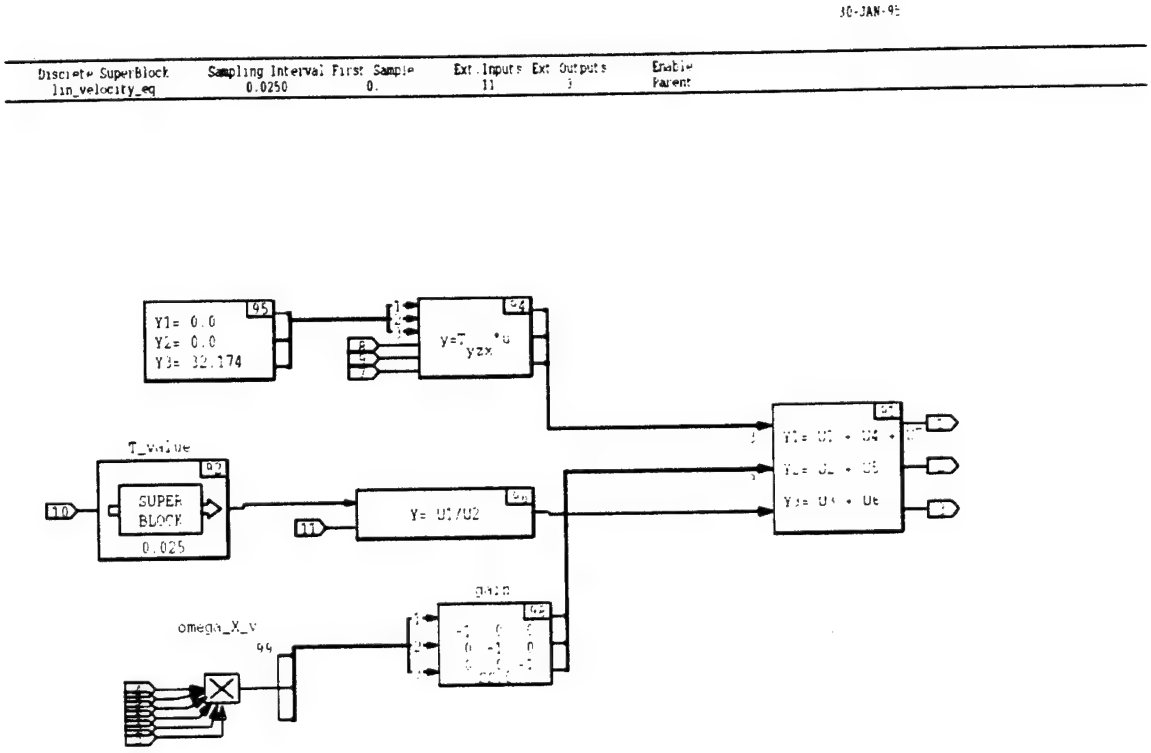


Figure A.22: 'lin_velocity_eq' SuperBlock

30-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext. Inputs	Ext. Outputs	Enable
pitch_command	0.0400	0.	4	2	Parent

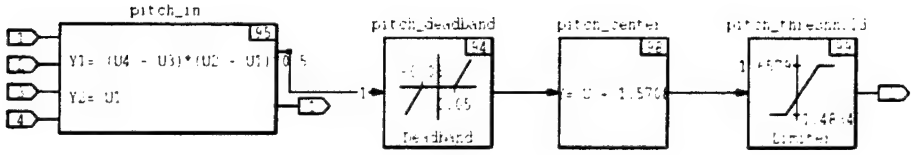


Figure A.23: 'pitch_command' SuperBlock

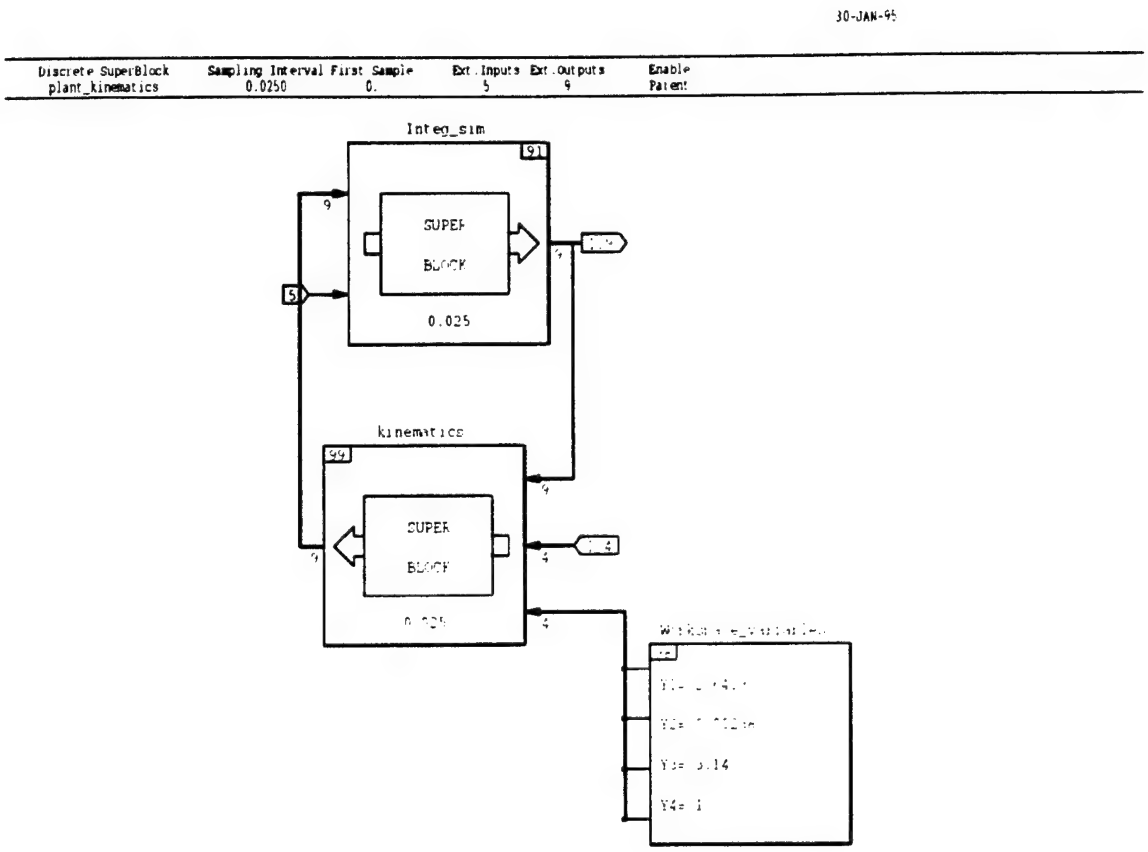


Figure A.24: 'plant_kinematics' SuperBlock

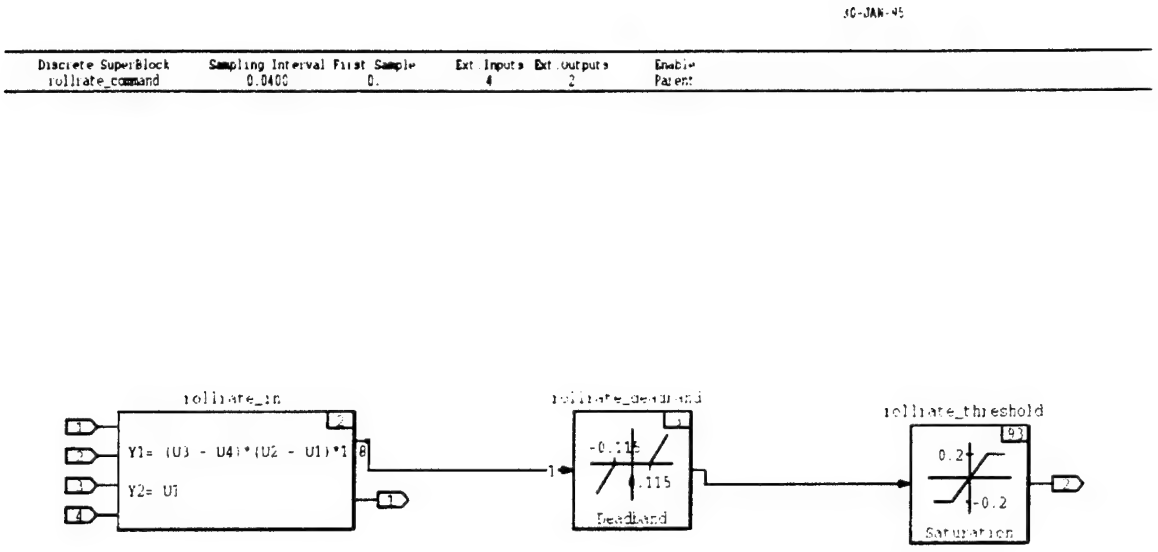


Figure A.25: 'rollrate_command' SuperBlock

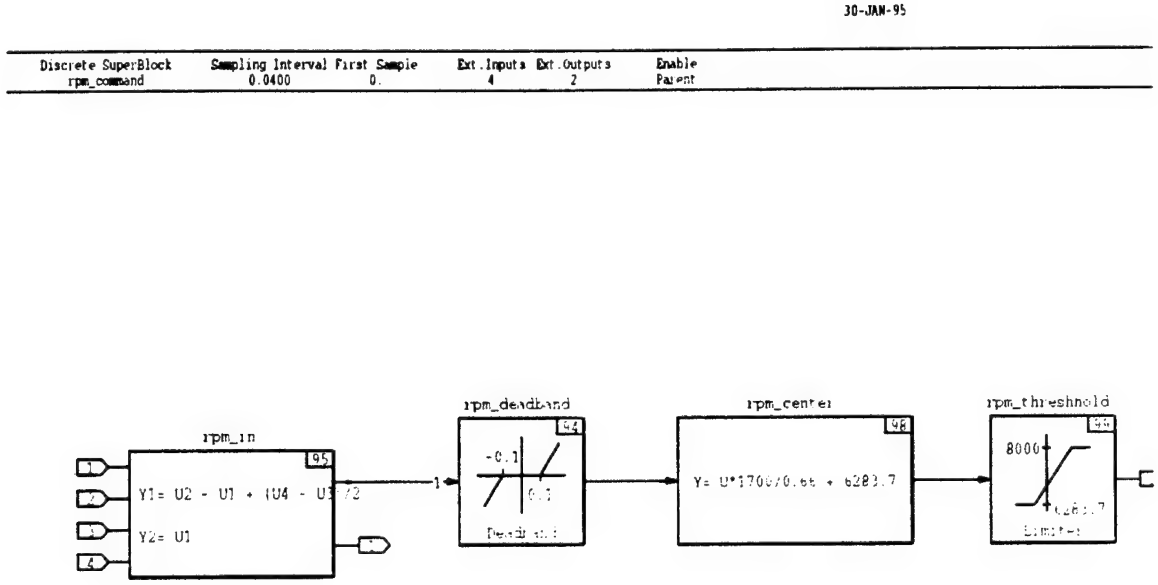


Figure A.26: 'rpm_command' SuperBlock

31-JAN-95

Discrete SuperBlock	Sampling Interval	First Sample	Ext.Inputs	Ext.Outputs	Enable
T_value	0.0250	0.	1	1	Parent

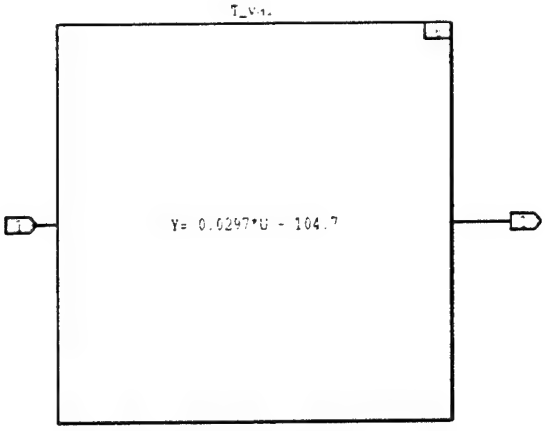


Figure A.27: 'T_value' SuperBlock

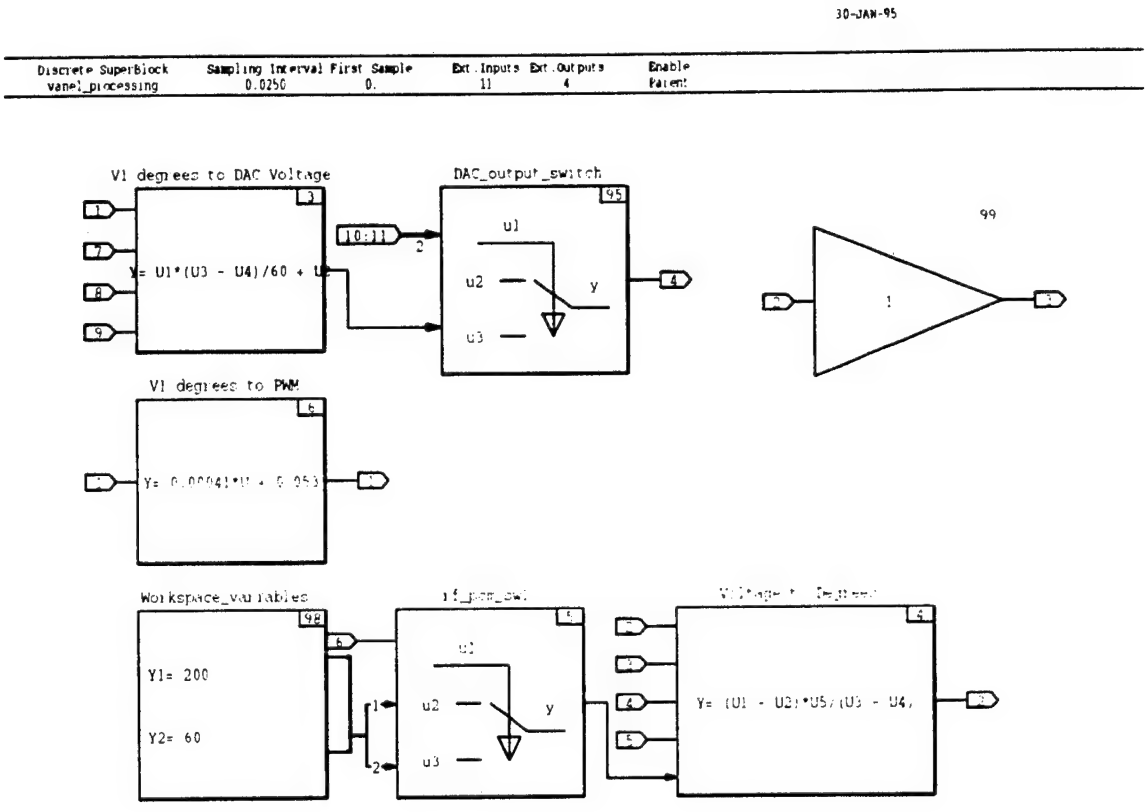


Figure A.28: 'vane1_processing' SuperBlock

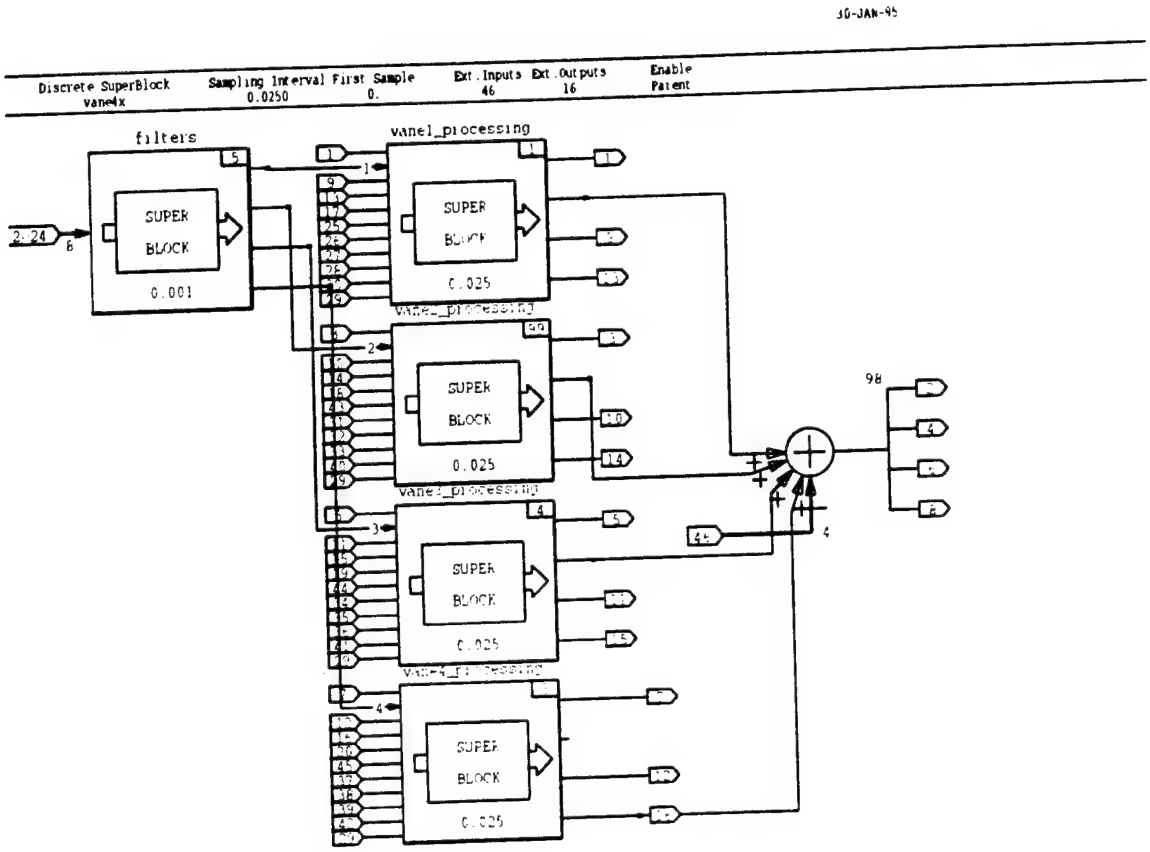


Figure A.29: 'vane4x' SuperBlock

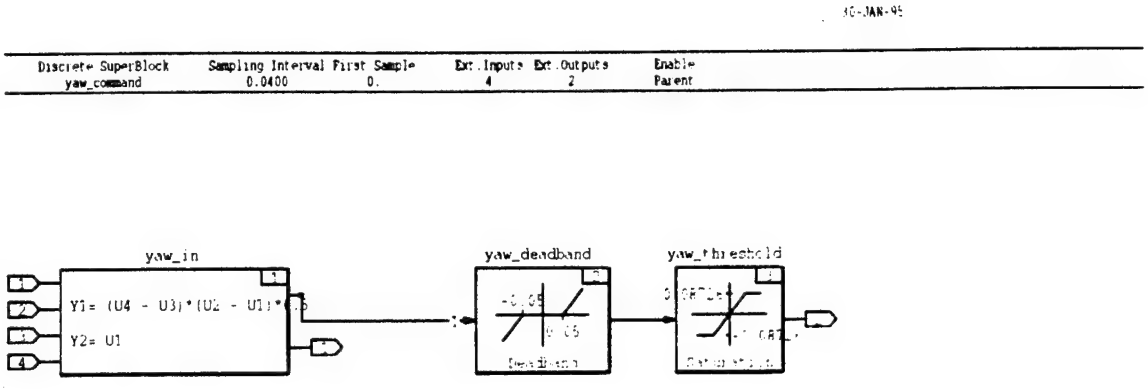


Figure A.30: 'yaw_command' SuperBlock

APPENDIX B. SERIAL DRIVERS FOR IMU & AC100C30 INTERFACE

This Appendix contains the procedure for developing serial drivers for the AC100C30 and provides instructions on how to convert hexadecimal IMU data to integer values. Finally it includes the C code used in the serial drivers for the interface of the IMU with the AC100 Model C30. The drivers consist of the part provided by ISI ('user_ser.c') and the part that was user modified to meet the IMU-controller integration requirements. Codes for both the hexadecimal and the binary output formats are provided.

A. SERIAL DRIVER DEVELOPMENT

The first step is to add the desired inputs and outputs to the SystemBuild model. The real time code '.rtf' file should then be generated, as described in the ISI manuals [Ref. 18]. Next any additional IA modules and connections should be created as desired. Finally the Hardware Connection Editor (HCE) needs to be modified, so that the appropriate IP_SERIAL modules and channels for the applicable external inputs and outputs are selected.

Each serial port can have both inputs and outputs, corresponding to transmitting and receiving. It is good practice, to keep a module dedicated to input or output, but not both, if the application allows it. Care should be taken so that the software serial channels selected in the "ch #" column of the HCE, correspond to the order in which the SuperBlock external inputs and outputs will be sent to the IP_SERIAL software drivers [Ref. 6].

The next step is to include the file **'user_ser.c'** provided by ISI, in the working directory and modify it as needed for the application. This file contains the 'get_SERIAL_parameters', 'user_SERIAL_out' and 'user_sample_SERIAL_in' functions, that should be modified to suit the application. Each function is called once on each sampling cycle, for each port, on each serial module. This file is a template used for modifications to suit a particular user application and can be found in the '\ac100c30\ipserial\serial' directory on the 'ac100' or 'america' 486 PC's [Ref. 6]. The three functions included in the 'user_ser.c' file, are presented below.

1. The 'get_SERIAL_parameters' Function

This function configures the serial ports for use, setting parameters such as baud rate, parity, stop bits, etc., and will be called once, for each channel and module used. The buffer size is also determined by this function. If different parameters need to be set for different ports (i.e., A or B on a given module), the

'hardware_channel' parameter in the function can be tested using an if( ) statement. This function argument will take on the value of 1 for port A, and 0 for port B. For example, if it is desired to set different baud rates for port A and B, this function could contain two if( ) statements. One should check if 'hardware_channel' is equal to 1, and if so, set the appropriate parameters for port A. The next step would be to check if 'hardware_channel' is equal to 0, and if so, set the appropriate parameters for port B. Likewise, if different parameters need to be set for different IP_SERIAL modules, the value of the 'IOdevice' structure argument 'device->config.module', should be checked using an if( ) statement. The value of 'device->config.module' will be equal to the number of the IP_SERIAL module, that the function is being called for. The 'IOdevice' structure is defined in the 'iodriver.h' and 'iodefs.h' header files, and should not be tampered with, other than to check the value of the module number. [Ref. 6]

Related to the 'get_SERIAL_parameters' function are error messages generated by an unexpected condition in the serial I/O of the C30. The most common of these messages are listed below with their explanation.

- "Serial channel 0 encountered a framing, parity, or overrun error": an undocumented error message that occurred when the baud rate of transmitting and receiving ends did not match.
- "C30 has died or is in an infinite loop": encountered when the data sent or received, caused the real time simulation to have a numerical error.
- "Serial channel 0 has a ring buffer overflow": a frequently encountered error message affiliated to ring buffer size. Occurred if the buffer is smaller than the number of bytes sent to it per cycle, or if the transmitting end is sending data to the IP_SERIAL port faster than it is being read. The error message showed up both on the C30 computer screen and in the command window, from which the AC100 GUI was run. At the same time a continuous beeping sounded from the GUI workstation. Nevertheless, these audio and visual error messages stop after a while, and once the specific application is started, it functions without problems.

2. The 'user_SERIAL_out' Function

This is the serial driver function for writing data to the serial port. In this function, the data can be formatted as desired, then written to the serial port as an unsigned character array, using the driver function 'write_serial'. The external output states of the model come into 'user_SERIAL_out' as a floating point array called 'model_float[]'. The states are arranged in the order selected for the channel numbers (ch# column) in the HCE. These values can be scaled, modified, or transformed as desired, using C code, and must then be placed as individual bytes into an unsigned character array. This array is later passed to the function 'write_serial' , which puts the data in the serial buffer, and transmits it with the selected serial communication parameters. [Ref. 6]

One difficulty in placing the values as individual bytes in the unsigned character array, is the fact that the compiler for the C30 uses 4 bytes (32 bits) for all data types (signed character, integer, long integer, or floating point number). Additionally, Texas Instruments that makes the C30, uses a non-IEEE standard floating point number representation, so a float on the C30, is not equivalent to an IEEE float on the receiving end. The best way to solve this problem, is to send the floating point values either as a string, or to scale them to a long integer. The disadvantage of the string method is that no conversion from float to string exists in the standard libraries, with either the C30 compiler, or other standard PC compilers, even though string-to-float exists. The other difficulty with this method is the variable string length, which depends on the number of digits carried. The only problem with the scaling to a long integer, is to properly define a scaling array with suitable scale values for the expected ranges of each variable. This was the method chosen. [Ref. 6]

To convert the scaled long value to an array of unsigned characters, a C language union was used. The union allocates only as much memory as its largest member, and then all the members share that memory. So, if a union contains both a long integer (which takes up four bytes in memory) and a four byte unsigned char array, only four bytes will be allocated. The long can then be assigned to the union, and pulled back out as four unsigned character bytes. The syntax for the union is [Ref. 6]

```
/*this structure joins 4 unsigned character bytes into one data type*/
typedef struct _bytes
{
        unsigned byte1 :8;
        unsigned byte2 :8;
        unsigned byte3 :8;
        unsigned byte4 :8;
} _bytes;
/*this union merges a _bytes structure and a long int into the same memory location */
typedef union longbuf
{
        long buf;
        _bytes bytes;
} longbuf;
```

The syntax in the 'structure_bytes' sets up four 8 bit bytes, grouped as a four byte word, which is then incorporated in a union with a long integer. A variable of type 'longbuf' can then be declared and

assigned to, as a long. Next, each byte of the union's bytes member should be assigned to the unsigned char array, that will pass to 'write_serial'. The reverse logic can be used on the receiving end, to convert back from the received unsigned character array to a long integer. This method is used in several places in the 'user_ser.c' files for the IMU, presented in Appendix B. [Ref. 6]

3. The 'user_sample_SERIAL_in' Function

This function allows the user to read data from the serial port, and then modify the data as desired, before writing it to the 'model_float[]' array. This array represents the SystemBuild external inputs, which are also in the order selected by the channel numbers (Ch# column) in the SB INPUTS screen of the HCE. The 'read_serial' function reads a specified number of bytes from the serial buffer, into a function argument unsigned character array. These bytes can then be transformed back into the data they represent, by conversions similar to those discussed above, for sending the data. There is also a checking if( ) statement in this function, which makes sure the appropriate number of bytes has arrived in the buffer before reading. If they have not, a global counter ('usr_ptr->update_count') is incremented, and this counter represents the number of clock cycles, that have passed with no update to the external serial inputs. If the number of clock cycles since the last data, exceeds a user defined value (usr_ptr->update_interval' which is set in 'get_SERIAL_parameters' function), the function will dispatch an error message and attempt to reset the serial port. [Ref. 6]

If separate handling of outputs is required for different serial channels or modules, if( ) statements can be added to check the values of 'ser_channel' or 'device->config. module' and then do the proper formatting in the 'get_SERIAL_ parameters' function section. It should be noted that, the ser_channel values are 0 for port B and 1 for port A, and the module corresponds to the one selected in the HCE for the appropriate outputs.

Modifications to the above three functions are necessary prior to proper operation of the IP_SERIAL module. The final step after setting up the 'user_ser.c' file to process the incoming and outgoing data, is to place the following lines in the 'sa_user.cmd' file in the project directory :[Ref. 6]

```
COMPILE user_ser.c
LINKWITH user_ser
```

This will tell the apbuild utility to transfer this file to the project directory on AC100, and compile and link it with the project. If 'user_ser.c' calls any data conversion functions that are in separate C files, those files should have their own COMPILE and LINKWITH lines in 'sa_user.cmd' prior to the ones for 'user_ser.c' , as previously discussed. [Ref. 6]

B. IMU OUTPUT CONVERSION

To manipulate the hexadecimal output of the 'New' IMU and extract integer values, perform the following:

- Convert the heading and altimeter hexadecimal numbers directly into an integer which is always positive. If the number is in two's complement format, first convert the 4 bytes to signed hexadecimal. If the number is between 0000H and 7FFFH. do nothing. If the number is between 8000H and FFFFH, subtract the value from 1000H and make it a negative value. This has the same effect as negating the value.
- Next, convert the number from hexadecimal to signed integer value. Then multiply the number by the appropriate full F.S. range, and divide by the corresponding F.S. hexadecimal value (see Table 3.2).

The conversions are done as follows (the values must first have been converted from hexadecimal to signed integer):

Bank (°) = Bank(Integer)*180°/32768

Elevation (°) = Elevation (Integer)*90°/32768

Heading (°) = Heading (Integer)*360°/65536

X Acc (g's) = X Acc (Integer)* 10 g's/32768

Y Acc (g's) = Y Acc (Integer)* 10 g's/32768

Z Acc (g's) = Z Acc (Integer)* 10 g's/32768

Roll Rate (°/second) = Roll Rate (Integer) * 100°/sec/32768

Pitch Rate (°/second) = Pitch Rate (Integer) * 100°/sec/32768

Yaw Rate (°/second) = Yaw Rate (Integer) * 100°/sec/32768

Altimeter Input = Altimeter (Integer) * 5 VDC/65536

Manipulation of the hexadecimal output for the 'Old' IMU is performed as described above for the 'new' IMU unit, with the difference that the total length is now 38 bytes, while the appropriate F.S. values have to be used (see Tables 3.3 and 3.4).

C. SERIAL DRIVERS FOR THE IMU

1. Serial Code Part Provided By Isi

```
/*****************************************************************
**@ File      : user_ser.c
**@ Project   : turnover
**@ Edit level : 4
**@
**@ Abstract:  : File contains functions which the user
**@          must define to interface with IP-SERIAL device
**@          driver.
**@          The functions must be called
**@
**@          get_SERIAL_parameters
**@          user_poll_SERIAL_out
**@          user_sample_SERIAL_in
**@
**@          Templates for the functions are provided.
**@
**@          get_SERIAL_parameters
**@            Function sets the asynchronous communication
**@          parameters for the IP-SERIAL module.  Ring buffer
**@          sizes used to store received data must also be
**@          specified.
**@
**@          user_SERIAL_out
**@            Function is called with the floating-point
**@          model output vector for the current sampling interval.
**@          The user is responsible for creating the transmit
**@          byte array and calling function write_serial which
**@          transmits the byte array across the serial channel.
**@
```

```
**@          user_sample_SERIAL_in
**@            Function is called every sampling interval.  The
**@          user is responsible for filling the floating-point
**@          vector which is used as input to the model for
**@          the current sampling interval.
**@            Function has been modified as described in
**@          Modifications section below.
**@
**@
**@ Modifications:
**@ -----------------------------------------------------------
**@    Creation : 7-1--93  Henry Tominaga
**@    Revised  : 8-23-93  Brent Roman
**@    Revised  : 11-18-93 Steve Lynch
**@    Revised  : 05 June 1994  Brian Noyes
**@    Revised  : 10-23-94 David R. Kuechenmeister
**@           Modified the user_sample_SERIAL_in function to read data
**@     from the IMU-600D. Reads the 36 byte message from the IMU and returns
**@     the model_float array to the system build model. More details are in
**@     the function header.

*/
#include <stddef.h>
#include <stdlib.h>
#include <string.h>
#include "sa_types.h"
#include "types.h"
#include "ioerrs.h"
#include "errcodes.h"
#include "iodriver.h"
#include "ipserial.h"
#include <printx.h>

#include "hexint.h"
```

```
#define NULL 0
#define MESSAGE_SIZE 36

 /* semaphores and serial parameters for each physical channel */
private struct
{ boolean  allSent; boolean  broken; unsigned baud;} line_state[2];

struct user_type
{
   int update_interval;
   int update_count;

};


/* Define the types required to read the serial card */

unsigned char buffer_data[40];
int         index;

boolean first_frame = true;
float last_float[9];

/*****************************************************************
*****************************************************************
@  THE INTERNALS OF THE FOLLOWING THREE FUNCTIONS MUST
@  BE PROVIDED BY THE USER.  PLEASE TAILOR FOR YOUR
@  OWN SPECIFIC APPLICATION.
*****************************************************************
*****************************************************************/
/*****************************************************************
 Please define serial communication parameters and ring buffer
 sizes in this function.
```

```
The communication parameters that must be set are the
standard asynchronous (RS-232) communication parameters.

The parameters are
  parity : NONE, EVEN, or ODD;
  baud_rate : enter any standard baud rate;
  stop_bits: ONE, TWO, or ONE_AND_HALF;
  transmit_data_size: 5, 6, 7, or 8 (bits);
  receive_data_size: 5, 6, 7, or 8 (bits);
  clock_multiplier: 1, 16, 32, or 64.
 Please follow syntax given in the template function.
 Please set parameters for the channels used.

 Please also set the ring buffer size to be used for the
 receive buffer.  The size should exceed the maximum
 anticipated number of bytes that will arrive at the
 IP-Serial Module during one sampling interval.
*****************************************************************/
public void get_SERIAL_parameters
  (unsigned int              hardware_channel,
   volatile struct user_param *device_param,
   volatile struct ring_buffer_param *rec_buffer,
   IOdevice                  *device)
{
 struct user_type *user_ptr;
 serial_param_type *serptr;

 serptr = device->parameters;

 if (SERIAL_USER_PTR == NULL)
 {
   SERIAL_USER_PTR = (struct user_type *)malloc(sizeof(struct user_type));
   user_ptr      = SERIAL_USER_PTR;
   user_ptr->update_interval  = 1000;
```

```
  user_ptr->update_count    = 0;
}


if (hardware_channel == chanA || hardware_channel == chanB)
{
 /* set parameters for channel A (IP-SERIAL channel 1)*/
 device_param->parity          = NONE;
 device_param->baud_rate       = 9600;
 device_param->stop_bits       = ONE;
 device_param->transmit_data_size = 8;
 device_param->receive_data_size  = 8;
 device_param->clock_multiplier   = 16;
 /* set size for receive ring buffer */
 rec_buffer->buffer_size       = 1000;
}
else
{
 printx("INVALID CHANNEL\n");
}

}
/* get_SERIAL_parameters */

/*************************************************
@ Function : user_SERIAL_out
@ Abstract : Please specify user-defined serial output
@            function here.  Function called as
@            scheduled output event.
@
@ Inputs   : device     (user-transparent pointer to
@                        system attributes; DO NOT MANIPULATE)
@            model_float[] (SystemBuild output vector)
@            ser_channel (serial channel)
```

```
@
@ Outputs :
@
@ Returns : error status
@
*************************************************/
typedef struct _bytes
{
 unsigned byte1 :8;
 unsigned byte2 :8;
 unsigned byte3 :8;
 unsigned byte4 :8;
} _bytes;

typedef union longbuf
{
        long buf;
        _bytes bytes;
} longbuf;

public RetCode user_SERIAL_out(IOdevice *device,
                               float model_float[],
                                       u_int ser_channel)
{
        Byte      cbuffer[50], buff2[10];
        u_int     i,j;
        int               k;
        longbuf  data;
        /* Scale factors for fuel, body angle, x vel, z vel, x pos, x accel,
                                       z pos, z accel, ang vel ang accel    */
        float scale[] = {163.835, 3276.7, 3276.7, 3276.7, 327.67, \
                         10922.33, 16383.5};
```

```
/*************************************************************
* Given floating point model output, scale the numbers      *
* to an integer (quantized based on maximum state           *
* values), and then convert them to a four digit hex        *
* numbers. Put these values in a buffer, and send as a      *
* 29 character, carriage return terminated string.          *
*************************************************************/
if (numbytes_in_buffer(device->parameters) == 0)
{
        for (i = 0; i < 7; i++)
        {

                k=model_float[i]*scale[i];
                itohex(k,buff2);
                for (j=0;j<4;j++) cbuffer[4*i+j] = buff2[j];

        /*************************************************************
        * Fill the output buffer with data to be transmitted        *
        * by the background portion of the serial driver            *
        *************************************************************/
         }
         cbuffer[28]='\r';
}
return OK;
} /* user_SERIAL_out */
```

Next the user modified part of the code must be appended.

D. CODE LISTING FOR HEX OUTPUT MODES

Appended after the code provided by ISI.

```
/*************************************************
@ Function : user_sample_SERIAL_in
@ Abstract : Please specify user-defined serial input
@          function here. Function called as
@          scheduled input event.
@
@ Inputs   : device     (user-transparent pointer to
@                    system attributes; DO NOT MANIPULATE)
@          ser_channel (serial channel)
@
@ Outputs  : *model_float (pass back floating pt vector)
@
@ Returns  : error status
@
@
@ Modifications:
@        Tailored the function to read the IMU 600D outputs. The IMU outputs
@        data at 9600 bps in 36 byte messages followed by a <CR> and a <LF>.
@        On the first frame, the buffer is read until a <CR> is located, then
@        the following <LF> is also read and thrown away. Now the message is
@        in sync with the required message format. On succeeding frames, the
@        serial buffer is read until the last <CR> is encountered, then the
@     message is processed into the model float array expected by the model.
@
@ Variables       Scope   Duration  For 38    For 42
@ buffer_data      file    fixed      [40]      [50]  ** Can vary
@ index            file    fixed
@ MESSAGE_SIZE     file    fixed      36        40
@ first_frame      file    fixed
@ last_float       file    fixed      [9]       [10]
```

```
@
@ item          local   auto
@ linefeed      local   auto
@ k,
@ scale,
@ default_data    local   auto      [9]      [10]
@
*****************************************************/
public RetCode user_sample_SERIAL_in(IOdevice *device,
                                     float model_float[],
                                     u_int ser_channel)
{
unsigned char item[1],
        linefeed[1];

struct user_type *user_ptr;
        /* Scale factors for A_x, A_y, A_z,
                          p,  q,  r,
                          heading, bank, roll*/

float  scale[] = {-3.0/32768.0, -3.0/32768.0, -3.0/32768.0,
                  100.0/32768.0, 100.0/32768.0, 100.0/32768.0,
                  -180.0/32768.0, 60.0/32768, 60.0/32768.0 },
     default_data[] = { 0.0, 0.0, -1.0, 10.0, 10.0, 10.0,
                0.0, 10.0, 10.0};

        /*************************************************************
        * set user pointer to buffer allocated by get parameters  *
        * this buffer is passed around with the structure device  *
        * and should only be accessed via the SERIAL_USER_PTR     *
        * define                                  *
        *************************************************************/
user_ptr      = SERIAL_USER_PTR;
```

```
      /*********************************************************
      * In this model we are expecting 38 bytes of data.      *
      * Data is processed in the function process_buffer_data *
      * Scaling, based on the scale[] is also completed in that*
    * function.                                               *
      *********************************************************/

if (first_frame){  /* Remove all characters up to the first <CR>,
                      then strip off the following <LF>
                      Message is now in sync with the expected
                      format  */
  index = 0;

  do{
    read_serial(device->parameters,1,item);
  } while(item[0] != '\r');
  read_serial(device->parameters,1,linefeed);

  model_float = default_data;
  index = 0;

  first_frame = false;
}


else{
  while( (numbytes_in_buffer(device->parameters)) > (MESSAGE_SIZE + 2) ) {
      /*****************************************************************
      check to see if the number of bytes in the serial buffer is
          greater than a complete message size, including the <CR> and <LF>.
      when it is that large, read the contents of the serial buffer into
          the local buffer. As soon as a <CR> is detected, process the
          contents of the local buffer into the model_float array.
      *****************************************************************/
```

```
for( read_serial(device->parameters,1,item);
    item[0] != '\r';
    read_serial(device->parameters,1,item)){
  buffer_data[ index ] = item[0];
  index++;
  } /* end for loop */


/* when a carriage return is found, strip the linefeed */
read_serial(device->parameters,1,linefeed);

if (index >= (MESSAGE_SIZE) ) {   /* proper size message */
  return_model_data( buffer_data,
                scale,
                model_float );
  index = 0;

}
else if (index < (MESSAGE_SIZE) ){  /* message too small */
  index = 0;
  model_float = last_float;
  IOerr(SERIAL_read_error,ser_channel);
} /* end if */

} /* end while loop */

/**************************************************************************This section only
executes if there are no bytes in the buffer
**************************************************************************/
  if ( numbytes_in_buffer(device->parameters) == 0) {
    model_float = default_data;
    /* check to make sure not too many cycles go by without
      an update to model_float.
```

```
      */
      user_ptr->update_count++;
      if ( user_ptr->update_interval < user_ptr->update_count){
        IOerr(SERIAL_line_break,ser_channel);
        reset_buffer(device->parameters);
        user_ptr->update_count = 0;
      } /* end if */
    } /* end if;*/


  } /* end if */

return OK;

} /* user_sample_SERIAL_in */



/*****************************************************************************
return_model_data will do the conversion from buffer data to
*       the model_float array with a call to the function hextoi.
*       To use this function with the IMU that sends 10 word messages,
*     simply change the counter index y from 9 to 10. The scale array
*       should also be updated for 10 elements and the differing order
*****************************************************************************/

void return_model_data(unsigned char *buffer_data,
                float   *scale,
                float   model_float_temp[]
                ){

unsigned char four_byte_buffer[4];
int     y, z;
```

```
  for ( y=0; y<9; y++){

    for(z=0;z<4;z++){
      four_byte_buffer[z] = buffer_data[y*4+z];
    } /* end for loop */

    model_float_temp[y] = ((float)hextoi(four_byte_buffer))*scale[y];
    last_float[y] = model_float_temp[y];

  } /* end for loop */
} /* end return_model_data */
```

E. CODE LISTING FOR BINARY OUTPUT MODE

Appended after the code provided by ISI.

```
/************************************************
@ Function : user_sample_SERIAL_in
@ Abstract : Please specify user-defined serial input
@          function here. Function called as
@          scheduled input event.
@
@ Inputs   : device     (user-transparent pointer to
@                    system attributes; DO NOT MANIPULATE)
@          ser_channel (serial channel)
@
@ Outputs  : *model_float (pass back floating pt vector)
@
@ Returns  : error status
@
@
@ Modifications:
@        Tailored the function to read the IMU 600D outputs. The IMU outputs
@        data at 9600 bps in 20 byte messages followed by a <CR>.
@        On the first frame, the buffer is read until a <CR> is located
@        Now the message is
@        in sync with the required message format. On succeeding frames, the
@        serial buffer is read until the last <CR> is encountered, then the
@     message is processed into the model float array expected by the model.
@
@ Variables       Scope   Duration  Binary
@ buffer_data     file    fixed     [25]  ** Can vary
@ index           file    fixed
@ MESSAGE_SIZE    file    fixed     20
@ first_frame     file    fixed
@ last_float      file    fixed     [10]
```

```
@
@ item          local   auto
@ linefeed      local   auto
@ k,
@ scale,
@ default_data    local   auto      [10]
@
*****************************************************/
public RetCode user_sample_SERIAL_in(IOdevice *device,
                                     float model_float[],
                                     u_int ser_channel)
{
unsigned char item[1],
        linefeed[1];

struct user_type *user_ptr;
        /* Scale factors for bank, pitch, heading,
                A_x, A_y, A_z,
                        p,  q,  r,
                        height          */

float   scale[] = {180.0/32768, 90.0/32768, 360.0/65536,
            -10.0/32768, -10.0/32768, -10.0/32768,
             100.0/32768, 100.0/32768, 100.0/32768,
             5.0/65536},
     default_data[] = { 0.0, 0.0, 1.0,
                1.0, 1.0, 1.0,
                1.0, 1.0, 1.0,
                1.0 };

        /*********************************************************
        * set user pointer to buffer allocated by get parameters        *
        * this buffer is passed around with the structure device        *
        * and should only be accessed via the SERIAL_USER_PTR           *
```

```
        * define                                                    *
        ************************************************************/
user_ptr      = SERIAL_USER_PTR;

        /************************************************************
        * In this model we are expecting 38 bytes of data.          *
        * Data is processed in the function process_buffer_data     *
        * Scaling, based on the scale[] is also completed in that   *
        * function.                                                 *
        ************************************************************/

  if (first_frame){  /* Remove all characters up to the first <CR>,
                        then strip off the following <LF>
                        Message is now in sync with the expected
                        format  */
    index = 0;

    do{
      read_serial(device->parameters,1,item);
    } while(item[0] != '\r');

    model_float = default_data;
    index = 0;

    first_frame = false;
  }


  else{
    while( (numbytes_in_buffer(device->parameters)) > (MESSAGE_SIZE + 1) ) {
        /*****************************************************************
        check to see if the number of bytes in the serial buffer is
            greater than a complete message size, including the <CR>.
        when it is that large, read the contents of the serial buffer into
```

```
        the local buffer. As soon as a <CR> is detected, process the
        contents of the local buffer into the model_float array.
      ************************************************************************/

    for( read_serial(device->parameters,1,item);
       item[0] != '\r';
       read_serial(device->parameters,1,item)){
      buffer_data[ index ] = item[0];
      index++;
      } /* end for loop */


    if (index >= (MESSAGE_SIZE) ) {   /* proper size message */
      return_model_data( buffer_data,
                  scale,
                  model_float );
      index = 0;

    }
    else if (index < (MESSAGE_SIZE) ){  /* message too small */
      index = 0;
      model_float = last_float;
      IOerr(SERIAL_read_error,ser_channel);
    } /* end if */

  } /* end while loop */

/************************************************************************This section only
executes if there are no bytes in the buffer
************************************************************************/
    if ( numbytes_in_buffer(device->parameters) == 0) {
      model_float = default_data;
      /* check to make sure not too many cycles go by without
        an update to model_float.
```

```
    */
    user_ptr->update_count++;
    if ( user_ptr->update_interval < user_ptr->update_count){
      IOerr(SERIAL_line_break,ser_channel);
      reset_buffer(device->parameters);
      user_ptr->update_count = 0;
    } /* end if */
  } /* end if;*/


 } /* end if */

return OK;

} /* user_sample_SERIAL_in */



/*************************************************************************
return_model_data will do the conversion from buffer data to
*       the model_float array with a binary to integer conversion.
*
*************************************************************************/

void return_model_data(unsigned char *buffer_data,
                float   *scale,
                float   model_float_temp[]
                ){

unsigned char two_byte_buffer[4];
int       k, y, z;

  for ( y=0; y<10; y++){
```

```
    for(z=0;z<2;z++){
      two_byte_buffer[z] = buffer_data[y*2+z];
    } /* end for loop */

    /* The MSB in each byte is 1, so strip it by masking, then add the bytes.
      Then apply two's complement arithmetic to get the proper integer
      value for the simulation. Cast the integer to a float, then scale. */

    k = 128*((int)(buf[1]&0x7F))+((int)(buf[0]&0x7F));
    if (k > 8191) {
      k = k - 16384;
    }

    model_float_temp[y] = (float)k*scale[y];
    last_float[y] = model_float_temp[y];

  } /* end for loop */
} /* end return_model_data */

/*********************************************************************
            END OF FILE
*********************************************************************/
```

APPENDIX C. FLIGHT TEST EQUIPMENT SETUP PROCEDURE

The equipment used for flight testing of the ARCHYTAS UAV is as shown in Figures 6.4 and 6.6. The SUN SPARC2 workstation 'intrepid' was removed from the aero computer network and set up to drive the C30 carrier PC (portable 486PC named 'ac100'). In the stand-alone configuration, the workstation functions the same as when it was part of the normal network. The files in the workstation have been rearranged for easier operation. The MATRIX$_x$ and AC100 software are now in the /mnt/home/matrixx directory while the user accounts are in the /mnt/home/archytas directory. This Appendix contains the detailed directory structure of the stand-alone UNIX workstation, as well as the procedure required to remove the flight test components from the UAV lab.

A. DIRECTORY STRUCTURE

To log in the workstation 'intrepid', the passwords are as follows:

- For login name "root", the password is "rootbeer".
- For login name "user", the password is "archytas". This is the commonly used login name.
- There are no other accounts set up on intrepid at this time. See the lab technician about how to do that.
- In order to remotely connect to other machines on the network (telnet or rlogin), one can either use the IP address directly, i.e., the number 131.120..., or one can enter the IP address and host name into the /etc/hosts file. For example, for connection with the PC 'america' on the network, include the line:

'131.120.149.94 america.aa.nps.navy.mil'

The directories on the SPARC workstation 'intrepid' are set up as follows:

- **matrixx:** It is the directory where all the necessary software to run MATRIX$_x$ and AC100 is placed.
- **baseline:** This should be the most stable directory. This directory should contain only the code and MATRIX$_x$ diagrams that are fully operational. It should only be updated when there is sufficient reason to make a change to a configuration that works. Additionally, a **Documentation** subdirectory exists under *baseline* where additional copies of software drivers may be kept.
- **develop:** It is the directory where new projects are tested before considering their inclusion in *baseline*. After a successful test the principals involved in the ARCHYTAS project should consider whether they should move the new software to *baseline*.
- **individual:** A directory environment for individual development. After successful testing, new projects should be moved to *develop*. Users can create their personal working subdirectories here.

B. MOVING PROCEDURE

To remove the UNIX workstation 'intrepid' and the 486PC 'ac100' from the network, without interrupting the continuity of the system, the following steps need to be taken

1. Remove the "T" fitting from the 'ac100'.
2. Disconnect the cable connecting the "T"' to 'intrepid'. This currently has a plastic shield on it.
3. Remove the white cable from the brown adapter connected to 'intrepid' and hook it into the "T" that was just removed from the 'ac100'.
4. Put a terminator on the brown adapter, and connect a "T" with a terminator on the loose cable from the brown adapter.
5. Finally plug the "T" with the terminator into the 'ac100'. The workstation and 486PC are now disconnected from the network and are ready to operate in stand-alone mode for flight testing.

WARNING: Turning the SUN workstation OFF requires the following procedure:

- Logout from the current account.
- Login as "shutdown" with password "shutmedown"
- The shutdown procedure will initiate, and the screen will show "#" signs. Type "halt" at any point. Shortly after that, the procedure will be halted and the workstation can be turned OFF.

Neglectance to follow the described procedure will result in inability to run MATRIX$_x$ software.

APPENDIX D. SLQ-96 SPECIFICATIONS AND OPERATION

This Appendix contains the manufacturer's specifications and the operating instructions for the Repco SLQ-96 radio data link.

A. SPECIFICATIONS

SYSTEM

RF frequency range	138-174 MHz
	406-430, 450-512 MHz
	928-960 MHz
Baud rate	4800/9600 bps
Synchronous clocking	Internal or External
RF transmission mode	
138-174, 406-430,	FCC 16K0F1D
450-512 MHz	
928-960 MHz	FCC 12K0F1D
Operational modes	Simplex, half duplex, or full duplex in all frequency ranges
Operating temperature range	0° C to +60° C
Interface	RS-232, synchronous or asynchronous:
	(optional RS-422/485 interface with RS-530 pinout)
Link delay	17-20 msec

RF TRANSMITTER AND RECEIVER

Turn-on settling time	≤ 5 msec
Transmit frequency stability	
138-174 MHz	+/-5 ppm
406-512 MHz	+/-5 ppm
928-960 MHz	+/- 1.5 ppm
Transmit output power	2 or 4 W

POWER SUPPLY

Voltage	+12.5 V D C (+/-2 0 %)

Transmit current drain

4W	≤2 A
2W	≤1 A
Receive/standby current drain	300 mA

B. OPERATION AND CONFIGURATION

1. Input/Output Signals

Input/output signal connections for the SLQ-96 are made via a standard DB-25 (25-pin D) connector. The pinouts for the DB-25 are in Table D.1; all signals meet standard EIA, RS-232 specifications. Additional descriptions of each I/O signal and its function are also provided .

Pin	Function	Direction
1	Chassis ground	
2	TXD (transmit data)	To modem
3	RXD (receive data)	From modem
4	RTS (request to send)+	To modem
5	CTS (clear to send)	From modem
6	DSR (data set ready)	From modem
7	Signal ground	
8	DCD (data carrier detect)	From modem
15	Internal Tx clock*	From modem
17	Receive clock*	From modem
20	DTR (data terminal ready)+	To modem
24	External Tx clock*	To modem

* Used only for synchronous operation.

+ Handshaking signals that must be supplied by the DTE to operate in the full hardware handshaking mode (if DTR is not available from the DTE. DTR can be strapped to DSR to provide proper handshaking).

Table D.1: SLQ-96 I/O Signals

TXD (transmit data). Data sent from the DTE to the modem for transmission to a remote modem.

RXD (receive data). Data that has been received by the modem from a remote location and is subsequently being transferred back to the data terminal equipment from the modern.

RTS (request to send). Active only when operating in the full hardware handshaking mode (and not in the three-wire mode), RTS is a handshake signal from the DTE to the modem that informs the modem that the DTE needs to transmit data. The modem responds by keying the transmitter - if it is not already keyed as it would be in the continuous mode - and, after a timed delay of between 0-200 ms (switch-selectable at the modern), sending a CTS signal to the DTE.

CTS (clear to send). A handshake signal active only in the full hardware handshaking mode (and not in the three wire mode). CTS is sent from the modem to the DTE to inform the DTE that the modem is ready to transmit data. CTS must always be preceded by an RTS signal from the DTE to the modern. The delay between RTS and CTS can be set between 0-200 ms at the modem.

DSR (data set ready). DSR is supplied by the modem to the DTE whenever the modem is turned on and ready to receive data from the DTE.

DCD (data carrier detect). DCD is a handshaking signal that is active in the full hardware handshaking mode but not in the three-wire mode. The modem sends a DCD signal to the DTE whenever the modem receives an on-frequency transmission from another modern. The DTE responds with a DTR signal if it is ready to receive

Internal Tx Clock (synchronous operation only). When operating synchronously, the modem provides an internal clock signal to control the transmission of data to the DTE. (An external transmit clock signal from the DTE can be substituted if applied at pin 24).

Receive Clock (synchronous operation only). The modem provides n receive clock signal at pin 17 for synchronous clocking of data transferred between the modem and the DTE.

DTR (data terminal ready). DTR - another handshaking signal that is active in the full hardware handshaking mode but not In the three-wire mode - is sent from the DTE to the modem in response to a DCD signal from the modem. DTR indicates that the DTE is ready to receive data from the modem.

NOTE: If the DTE is not set up to supply a DTR signal to the modern, the DTR line can be strapped to the DSR line at the modem connector to provide proper handshaking.

External Tx Clock (synchronous operation only). If the modem's own clock is not used to provide transmit clocking, a clock signal from the DTE can be substituted if applied on pin 24.

2. Status Lights

Status lights, which monitor the status of the input/output lines, are located inside the case on the SLQ-96. A jumper (JS-17) on the processor board can be used to enable or disable the bank of lights ns desired. Brief descriptions of the function of each status light follow.

DSR (data set ready). The modem is ready to transmit or receive data.

DTR (data terminal ready). The data terminal is ready to receive data from the modem.

RTS (request to send). A signal to the modem that the data terminal has data to transmit.

CTS (clear to send). A signal to the data terminal that the modem is ready to receive data.

TXD (transmitting data). The transmitter is keyed and the modem is transmitting data.

RXD (receiving data). The modem is receiving valid data from another modem.

DCD (data carrier detect). The modem is receiving a radio signal that is the proper frequency but may or may not carry valid data.

TX (transmit). The modem's transmitter is keyed.

RX (receive). The modem's receiver is on.

TM (test mode). The modem is operating in the test mode (the test mode is selected using jumper JS-10 on the modem's processor board).

3. Dip Switch Settings

Three banks of DIP switches (a total of 24 switches) located on the figure board are used to configure a wide range of operating parameters. Table D.2 lists the switches and their functions and is followed by brief descriptions of all of the switch settings. Whenever a DIP switch is changed, the SLQ must be restarted for the change to take effect.

Note that functions such as I/O data rate, three-wire time out, CTS delay, and character length are determined by groups of switches rather than a single switch setting. For specific switch settings for these functions, refer to the individual switch descriptions.

NOTE: The DIP switch settings for the configuration used in the ARCHYTAS down link, are shown everywhere with bold characters.

I/0 Data Rate (SW2-2, 2-2). The I/O data rate (the rate of data transfer between the SLQ-96 and the DTE, not to be confused with the RF radio-to-radio data rate selected by SW3-8) can be set to 1200, 2400, 4800, or 9600 bps, according to the switch settings in Table D.3. If 9600 bps is selected, then SW3-8 must also be set to 9600 bps.

Three-Wire Time-out (SW2-4, 2-5). In the three-wire mode the modem ignores all handshaking signals to and from the DTE and keys the transmitter when the first character from the DTE is detected (just as if the RTS line was set true). To accommodate the delay required to stabilize the transmitter before transmitting data (approximately 9 milliseconds), the modem momentarily buffers the data from the DTE. The three-wire time-out duration (the delay between when the modem detects the last bit of data from the DTE and when it stops transmitting) is selected according to Table D.4.

NOTE: If the three-wire mode is disabled by placing both SW2-4 and SW2-5 in the *open* position, the modem operates

using full hardware handshaking.

CTS Delay (SW2-6, 2-7, 2-8). After receiving an RTS signal from the DTE, the modem waits a predetermined amount of time - the CTS delay - before sending a CTS signd back to the DTE. The length of the CTS delay is selected according to the switch positions shown in Table D.5.

Switch	Function
SW1-1 to SW1-8	Not used
SW2-1	I/0 data rate (to/from the DTE)
SW2-2	I/0 data rate (to/from the DTE)
SW2-3	Not used
SW2-4	Three-wire time-out/on-off
SW2-5	Three-wire time-out/on-off
SW2-6	CTS delay
SW2-7	CTS delay
SW2-8	CTS delay
SW3-1	**Half duplex (open)**, full duplex (closed)
SW3-2	Tx mode: switched (open), **continuous (closed)**+
SW3-3	Rx mode: **switched (open)**, continuous (closed)+
SW3-4	Stop bits: **one (open)**, two (closed)
SW3-5	Character length
SW3-6	Character length
SW3-7	**Asynchronous (open)**, synchronous (closed)
SW3-8	**9600 bps (open)**, 4800 bps (closed) select*

+The transmitting unit is set for continuous Tx and switched Rx, while the receiving unit is set for switched Tx and continuous Rx.

*Selects the rate of RF data transfer between SLQ-96 modems, not the I/O Data rate between the SLQ-96 and the DTE (see SW2-l, 2-2).

Table D.2: DIP Switch Functions

SW2-1	SW2-2	Description
Closed	Closed	1200 bps
Open	Closed	2400 bps
Closed	Open	4800 bps
Open	**Open**	**9600 bps**

Table D.3: Data Rate Selection

SW2-4	SW2-5	Description
Open	Open	three-wire mode disabled
Closed	**Open**	**5 millisecond time-out**
Open	Closed	100 millisecond time-out
Closed	Closed	500 millisecond time-out

Table D.4: Three-Wire Timeout Select

SW2-6	SW2-7	SW2-8	Description
Open	**Open**	**Open**	**0 millisecond delay**
Closed	Open	Open	5 millisecond delay
Open	Closed	Open	10 millisecond delay
Closed	Closed	Open	25 millisecond delay
Open	Open	Closed	50 millisecond delay
Closed	Open	Closed	100 millisecond delay
Open	Closed	Closed	150 millisecond delay
Closed	Closed	Closed	200 millisecond delay

Table D.5: CTS Delay Select

Half Duplex/Full Duplex (SW3-1). Half duplex operation (switch open) means that the modem cannot transmit and receive data at the same time, while full-duplex operation (switch closed) means that the modem can transmit and receive data simultaneously.

Tx Mode (SW3-2). When configured to the switched mode (switch open), the modem keys and then unkeys the radio transmitter whenever a data transmission is made. In the continuous mode (switch.closed) the tnmsmitter is keyed continuously - something that is usually done only with full-duplex modems.

Rx Mode (SW3-3). The Rx mode typically corresponds to the Tx-mode setting of the companion modem (the remote modem that the local modem is talking to). When the remote modem's transmitter is set to *switched,* the local modem's Rx mode should also be set to *switched* (this tells the local receiver to expect interruptions in the received signal and not to interpret these interruptions as data errors). On the other hand, if the remote modem's transmitter is set to *continuous,* the local modem will expect to continuouslly receive an RF signal and should, therefore, also be set to *continuous.*

Stop Bits (SW3-4). A stop-bits setting determines whether one or two bits are transmitted after each character to indicate the character's end. Most systems require one stop bit (switch open), although some require two (switch closed).

Character Length (SW3-5, 3-6). Character length of data words can be set to 6, 7, or 8 bits with even, odd, or no parity. The appropriate character length is determined by the data format used by the DTE connected to thc modem and is selected using the switch settings shown in Table D.6.

SW3-5	SW3-6	Description
Open	Open	Not used
Closed	**Open**	**8 bits (8 + none; 7 + parity)**
Open	Closed	7 bits (7 + none; 6 + parity)
Closed	Closed	6 bits (6 + none; 5 + parity)

Table D.6: Character Length Select

Asynchronous/Synchronous (SW3-7). Asynchronous or synchronous operation is selected according to the data-transfer format used to communicate with the DTE.

Asynchronous data transfer consists of a start bit and a stop bit at the beginning and end, respectively, of each data byte. These start and stop bits control data flow and eliminate the need for precise synchronization of the communicating data devices. Asynchronous operation is typical for most personal computers.

Synchronous data transfer - typically much faster than the asynchronous mode - relies on the synchronization of the transmitting and receiving devices, in this case the modem and the DTE. Precise timing devices in both units (transmit clocking can be supplied by either the modem or the DTE) are used to maintain synchronization. Synchronous transmission is often typical for terminals operating in a mainframe environment.

RF Data Transfer Rate (SW3-8). The rate of RF data transfer between radio moderns can be set to either 9600 bps (switch open) or 4800 bps (switch closed). This setting should not be confused with the I/O data rate, which is the rate of data transfer between the radio modem and the DTE and is controlled by switches SW2-l and 2-2. However, if the I/O data rate is set to 9600 bps, then the RF data transfer rate (SW3-8) also *must* be set for 9600 bps.

LIST OF REFERENCES

1 Stoney, R.B., *Design, Fabrication, and Test of a Vertical Attitude Take-Off and Landing Unmanned Air Vehicle,* Engineer's Thesis, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., June 1993.

2 Moran, Patrick J., *Control Vane Guidance for a Ducted-Fan Unmanned Air Vehicle,* Master's Thesis, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., June 1993.

3 Sivashankar, N., *Design, Analysis and Hardware-in-the-Loop Testing of a Controller for the Unmanned Aerial Vehicle ARCHYTAS,* Report on work done while a visiting scholar, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., August 1993.

4 Kuechenmeister, David R., *A Non-Linear Simulation For An Autonomous Unmanned Aerial Vehicle,* Master's Thesis, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., September 1993.

5 Moats, Michael L., *Automation of Hardware-in-the-Loop Testing of Control Systems for Unmanned Air Vehicles,* Master's Thesis, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., September 1994.

6 Noyes, B., *Hardware in the Loop Testing with the AC100C30 and Flight Management Unit,* Report on work completed for Directive Study Course AA 4900, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., June 1994.

7 Hoffman, Peter M., *Design and Synthesis of a Real-Time Controller for an Unmanned Air Vehicle,* Masters Thesis, Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA., September 1994.

8 Reichert, Frederick W., *Datalink Development for the Archytas Vertical Takeoff and Landing Transitional Flight Unmanned Aerial Vehicle,* Masters Thesis, Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA., June 1993.

9 Bess, Philip K., *Spread Spectrum Applications in Unmanned Aerial Vehicles,* Masters Thesis, Department of Electrical and Computer Engineering, Naval Postgraduate School, Monterey, CA., June 1994.

10 White, J.E., and Phelan, J.R., "Stability Augmentation for a Free Flying Ducted Fan," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, pp. 896-904, Monterey, CA., August 1987.

11 White, J.E., and Phelan, J.R., "Stability Augmentation and Control Decoupling for the Airborne Remotely Operated Device," *Journal of Guidance, Control and Dynamics*, vol. 14, no. 1, pp. 176-183, Monterey, CA., 1991.

12 Kress, G.A., *Preliminary Development of a VTOL Unmanned Air Vehicle for the Close Range Mission*, Master's Thesis, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., September 1992.

13 Slater, John M., *Theory of Inertial Navigation*, McGraw-Hill, New York, NY., 1964.

14 Kaminer, Isaac I., *AA 4276 - Avionics Systems Design*, Class Notes, Naval Postgraduate School, Monterey, CA., March 1994.

15 Lin, C., *Modern Navigation, Guidance, and Control Processing*, Prentice Hall, Atlanta, GA., 1991.

16 Watson Industries, Inc., *IMU-600D Owner's Manual*, August 1993.

17 Byerly, J., *Development of Equations-of-Motion in MATRIX$_X$ / SystemBuild*, Report on work completed for Directive Study Course AA 4900, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., June 1994.

18 Integrated Systems, Inc., *AC-100 Reference, Version AC3.3*, July 1993.

19 Integrated Systems, Inc., *AC-100 Model C30 Supplemental Reference, Version AC3.31*, July 1993.

20 Cibula, Andrew L., *Development and Testing of a VTOL UAV*, Masters Thesis, Department of Aeronautics and Astronautics, Naval Postgraduate School, Monterey, CA., March 1995.

21 Repco, Inc., *SLR-96 and SLQ-96 Service Manual*, April 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52 2
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Chairman, Code EC 1
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5121

4. Chairman, Code AA 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California 93943-5000

5. Dr. Isaac I. Kaminer, Code AA/Ka 2
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California 93943-5000

6. LCDR Michael K. Shields, Code EC/Sh 2
   Department of Electrical and Computer Engineering
   Naval Postgraduate School
   Monterey, California 93943-5100

7. Dr. Richard M. Howard, Code AA/Ho 1
   Department of Aeronautics and Astronautics
   Naval Postgraduate School
   Monterey, California 93943-5000

8. Dimitris E. Kataras 3
   Aiolou 24
   Ag. Paraskevi 15342
   Athens, GREECE